KARPAGAM ACADEMY OF HIGHER EDUCATION

Coimbatore-641 021

(For the candidates admitted from 2016 onwards)

DEPARTMENT OF COMPUTER SCIENCE, CA & IT

SUBJECT NAME : PROGRAMMING IN MATLAB

SEMESTER : III

SUBJECT CODE: 16CSU304B **CLASS: II B.SC CS**

Course Objective: A student who successfully completes this course should be able to learn how to use MATLAB, learn how to program in MATLAB, ability to create a computer program to solve problems in science and engineering.

Course Outcomes:

- To learn fundamental programming concepts using a block-structured language (MATLAB).
- To learn General problem-solving techniques, including the concept of step-wise refinement applied to the development of algorithms.

UNIT-I

Introduction to Programming: Components of a computer, working with numbers, Machine code, Software hierarchy.

UNIT-II

Programming Environment: MATLAB Windows, A First Program, Expressions, Constants, Variables and assignment statement, Arrays.

UNIT-III

Graph Plots: Basic plotting, Built in functions, Generating waveforms, Sound replay, load and save. Procedures and Functions: Arguments and return values, M-files, Formatted console input-output , String handling

UNIT-IV

Control Statements: Conditional statements: If, Else, Else-if, Repetition statements: While, for loop

UNIT-V

Manipulating Text: Writing to a text file, Reading from a text file, Randomising and sorting a list, searching a list. **GUI Interface**: Attaching buttons to actions, Getting Input, Getting Output

SUGGESTED BOOK

1. Amos Gilat. MATLAB: An Introduction with Applications(2nd ed). New Delhi: Wiley.

2. Stormy Attaway , 2009, Matlab: A Practical Introduction to Programming and Problem Solving, 2nd Edition, Butterworth Heinemann.

WEBSITES

1. http://oer.nios.ac.in/wiki/index.php/COMPUTER_AND_ITS_COMPONENTS
2. https://en.wikipedia.org/wiki/MATLAB
3. https://en.wikipedia.org/wiki/M_code
4. http://faculty.washington.edu/lum/website_professional/matlab/tutorials/Matlab_Tutorial_Beginner/matlab_tutorial_beginner.pdf
5. https://in.mathworks.com/help/matlab/learn_matlab/expressions.html
6. https://in.mathworks.com/products/matlab/choosing_hardware.html

ESE MARKS ALLOCATION

S.No	Category	Marks
1.	**Section A** 20 X1 = 20 Online Examination	20
2.	**Section B** 5x 2 =10	10
3.	**Section C** 5 x 6 = 30 Either 'A' or 'B' Choice	30
	Total	**60**

KARPAGAM ACADEMY OF HIGHER EDUCATION

(Coimbatore-641 021

(For the candidates admitted from 2016 onwards)

DEPARTMENT OF COMPUTER SCIENCE, CA & IT

STAFF NAME: D.MANJULA

SUBJECT NAME: PROGRAMMING IN MATLAB **SUB.CODE: 16CSU304B**

SEMESTER: III **CLASS : II B.SC CS**

LECTURE PLAN

S.No.	Lecture Duration (Period)	Topics to be Covered	Support Materials
Unit – I			
1.	1	Components of Computer	W1,W6
2.	1	Working with numbers	W2
3.	1	Machine Code	W3
4.	1	Software hierarchy	W1
5.	1	Matlab Architecture	W1
6.	1	Recapitulation and Discussion of important questions	
Total No. of Hours Planned for Unit-I			**6**
WEBSITES		W1 : http://oer.nios.ac.in/wiki/index.php/computer_and_its_components W2 :https://en.wikipedia.org/wiki/MATLAB W3: https://en.wikipedia.org/wiki/M_code W6: https://in.mathworks.com/products/matlab/choosing_hardware.html	
Unit – II			
1.	1	MATLAB Windows	S1:9–5, W3
2.	1	A First Program	W5
3.	1	Expressions	S2: 10-17, W5
4.	1	Constants	S2: 14, W4
5.	1	Variables	S1: 16-18
6.	1	Assignment statement	S2: 6-9, W5
7.	1	Arrays	S1: 35-55,S2: 30-31, W5
8.	1	Recapitulation and Discussion of important questions	
Total No. of Hours Planned for Unit-II			**8**
WEBSITES		**S1:** Amos Gilat. MATLAB: An Introduction with Applications(2nd ed). New Delhi: Wiley S2: Stormy Attaway , 2009, Matlab: A Practical Introduction to	

		Programming and Problem Solving, 2nd Edition, Butterworth Heinemann. W3: https://en.wikipedia.org/wiki/M_code W4: http://faculty.washington.edu/lum/website_professional/matlab/tutorials/Matlab_Tutorial_Beginner/matlab_tutorial_beginner.pdf W5: https://in.mathworks.com/help/matlab/learn_matlab/expressions.html	
Unit – III			
1.	1	Basic plotting, Built in functions	S1: 133-139, S1: 13-16, S2: 14-17, W5
2.	1	Generating waveforms	S2: 393-394
3.	1	Sound replay	W4
4.	1	Load	S1: 111-112, W4
5.	1	Save	S1:113, W4
6.	1	Procedure and Functions	S1: 219-244, W5
7.	1	Arguments	W5
8.	1	Return values	W5
9.	1	M-files	S1: 97-110
10.	1	Formatted console input-output	W5
11.	1	String handling	S1: 53-54, W5
12.	1	Recapitulation and Discussion of important questions	
		Total No. of Hours Planned for Unit-III	**12**
WEBSITES		S1: Amos Gilat. MATLAB: An Introduction with Applications(2nd ed). New Delhi: Wiley S2: Stormy Attaway , 2009, Matlab: A Practical Introduction to Programming and Problem Solving, 2nd Edition, Butterworth Heinemann. W4: http://faculty.washington.edu/lum/website_professional/matlab/tutorials/ W5: https://in.mathworks.com/help/matlab/learn_matlab/expressions.html	
Unit - IV			
1.	1	Conditional statements Representing Logical True and False	S1: 182-189, W5
2.	1	if Statement	S2: 82-86,W5
3.	1	if-Else Statement	S2: 87-88
4.	1	Nested if-Else Statements	S2: 88
5.	1	The Switch Statement. Menu Function	S1: 190-200,S2:93-96, W5
6.	1	Repetition statements For and Nested For	S2: 110-129,W2

7.	1	While and Multiple Conditions in while	S2:143-150
8.	1	Recapitulation and Discussion of important questions	
		Total No. of Hours Planned for Unit-IV	**8**
WEBSITES		S1: Amos Gilat. MATLAB: An Introduction with Applications(2nd ed). New Delhi: Wiley S2: Stormy Attaway , 2009, Matlab: A Practical Introduction to Programming and Problem Solving, 2nd Edition, Butterworth Heinemann. W2 :https://en.wikipedia.org/wiki/MATLAB W5: https://in.mathworks.com/help/matlab/learn_matlab/expressions.html	
Unit - V			
1.	1	Manipulating Text Writing to a text file	S2: 59-62,W4
2.	1	Reading from a text file	S2: 61-63,W1
3.	1	Randomising Sorting a list	S2: 372-378, W4
4.	1	Searching a list	S2: 382-392, W4
5.	1	GUI Interface Attaching buttons to actions	S2: 405-420, W4
6.	1	Getting Input	S2: 409-410,W4
7.	1	Setting Output	S2: 409-411,W4
8.	1	Recapitulation and Discussion of important questions	
9.	1	Recapitulation and Discussion of ESE question papers	
10.	1	Recapitulation and Discussion of ESE question papers	
11.	1	Recapitulation and Discussion of ESE question papers	
		Total No. of Hours Planned for Unit-V	**11**
WEBSITES		S2: Stormy Attaway , 2009, Matlab: A Practical Introduction to Programming and Problem Solving, 2nd Edition, Butterworth Heinemann. W1 : http://oer.nios.ac.in/wiki/index.php/computer_and_its_components W4: http://faculty.washington.edu/lum/website_professional/matlab/tutorials/	
		Total No. of periods	**45**

UNIT I

SYLLABUS

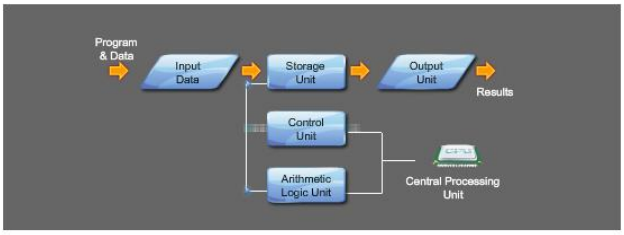
Introduction to Programming: Components of a computer, working with numbers, Machine code, Software hierarchy.

INTRODUCTION TO PROGRAMMING

Components of Computer

A computer system consists of mainly four basic units; namely input unit, storage unit, central processing unit and output unit. Central Processing unit further includes Arithmetic logic unit and control unit, as shown in the figure. A computer performs five major operations or functions irrespective of its size and make. These are

- ✓ it accepts data or instructions as input,
- ✓ it stores data and instruction
- ✓ it processes data as per the instructions,
- ✓ it controls all operations inside a computer, and
- ✓ it gives results in the form of output.



Functional Units:

a. Input Unit: This unit is used for entering data and programs into the computer system by the user for processing.

b. Storage Unit: The storage unit is used for storing data and instructions before and after processing.

c. Output Unit: The output unit is used for storing the result as output produced by the computer after processing.

d. Processing: The task of performing operations like arithmetic and logical operations is called processing. The Central Processing Unit (CPU) takes data and instructions from the storage unit and makes all sorts of calculations based on the instructions given and the type of data provided.

It is then sent back to the storage unit. CPU includes Arithmetic logic unit (ALU) and control unit (CU)

Arithmetic Logic Unit: All calculations and comparisons, based on the instructions provided, are carried out within the ALU. It performs arithmetic functions like addition, subtraction, multiplication, division and also logical operations like greater than, less than and equal to etc.

• Control Unit: Controlling of all operations like input, processing and output are performed by control unit. It takes care of step by step processing of all operations in side the computer.

Memory

Computer's memory can be classified into two types; primary memory and secondary memory

RAM

a. Primary Memory can be further classified as **RAM and ROM**.

• RAM or Random Access Memory is the unit in a computer system. It is the place in a computer where the operating system, application programs and the data in current use are kept temporarily so that they can be accessed by the computer's processor. It is said to be 'volatile' since its contents are accessible only as long as the computer is on. The contents of RAM are no more available once the computer is turned off.

ROM or Read Only Memory is a special type of memory which can only be read and contents of which are not lost even when the computer is switched off. It typically contains manufacturer's instructions. Among other things, ROM also stores an initial program called the 'bootstrap loader' whose function is to start the operation of computer system once the power is turned on.

b. Secondary Memory

RAM is volatile memory having a limited storage capacity. Secondary/auxiliary memory is storage other than the RAM. These include devices that are peripheral and are connected and controlled by the computer to enable permanent storage of programs and data.

- CD ROM

Secondary storage devices are of two types; magnetic and optical. Magnetic devices include hard disks and optical storage devices are CDs, DVDs, Pen drive, Zip drive etc.

- Hard Disk

Hard disks are made up of rigid material and are usually a stack of metal disks sealed in a box. The hard disk and the hard disk drive exist together as a unit and is a permanent part of the computer where data and programs are saved. These disks have storage capacities ranging from 1GB to 80 GB and more. Hard disks are rewritable.

Compact Disk

Compact Disk (CD) is portable disk having data storage capacity between 650-700 MB. It can hold large amount of information such as music, full-motion videos, and text etc. CDs can be either read only or read write type.

- Digital Video Disk

Digital Video Disk (DVD) is similar to a CD but has larger storage capacity and enormous clarity. Depending upon the disk type it can store several Gigabytes of data. DVDs are primarily used to store music or movies and can be played back on your television or the computer too. These are not rewritable.

Hard Disk

Input / Output Devices:

These devices are used to enter information and instructions into a computer for storage or processing and to deliver the processed data to a user. Input/Output devices are required for users to communicate with the computer. In simple terms, input devices bring information into the computer and output devices bring information OUT of a computer system. These input/output devices are also known as peripherals since they surround the CPU and memory of a computer system.

Input Devices

An input device is any device that provides input to a computer. There are many input devices, but the two most common ones are a keyboard and mouse. Every key you press on the keyboard and every movement or click you make with the mouse sends a specific input signal to the computer.

Keyboard

• **Keyboard**: The keyboard is very much like a standard typewriter keyboard with a few additional keys. The basic QWERTY layout of characters is maintained to make it easy to use the system. The additional keys are included to perform certain special functions. These are known as function keys that vary in number from keyboard to keyboard.

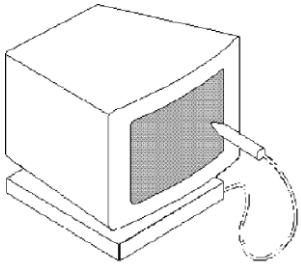
• **Mouse**: A device that controls the movement of the cursor or pointer on a display screen. A mouse is a small object you can roll along a hard and flat surface. Its name is derived from its shape, which looks a bit like a mouse. As you move the mouse, the pointer on the display screen moves in the same direction.

• **Trackball**: A trackball is an input device used to enter motion data into computers or other electronic devices. It serves the same purpose as a mouse, but is designed with a moveable ball on the top, which can be rolled in any direction.

• **Touchpad**: A touch pad is a device for pointing (controlling input positioning) on a computer display screen. It is an alternative to the mouse. Originally incorporated in laptop computers, touch pads are also being made for use with desktop computers. A touch pad works by sensing the user's finger movement and downward pressure. • Touch Screen: It allows the user to

operate/make selections by simply touching the display screen. A display screen that is sensitive to the touch of a finger or stylus. Widely used on ATM machines, retail point-of-sale terminals, car navigation systems, medical monitors and industrial control panels.

Light Pen: Light pen is an input device that utilizes a light-sensitive detector to select objects on a display screen.



• **Magnetic ink character recognition (MICR)**: MICR can identify character printed with a special ink that contains particles of magnetic material. This device particularly finds applications in banking industry.

• **Optical mark recognition (OMR)**: Optical mark recognition, also called mark sense reader is a technology where an OMR device senses the presence or absence of a mark, such as pencil mark. OMR is widely used in tests such as aptitude test.

• **Bar code reader**: Bar-code readers are photoelectric scanners that read the bar codes or vertical zebra strips marks, printed on product containers. These devices are generally used in super markets, bookshops etc.

Scanner

Scanner is an input device that can read text or illustration printed on paper and translates the information into a form that the computer can use. A scanner works by digitizing an image.

Output Devices:

Output device receives information from the CPU and presents it to the user in the desired from. The processed data, stored in the memory of the computer is sent to the output unit, which then converts it into a form that can be understood by the user. The output is usually produced in one of the two ways – on the display device, or on paper (hard copy).

•**Monitor**: is often used synonymously with "computer screen" or "display." Monitor is an output device that resembles the television screen (fig. 1.8). It may use a Cathode Ray Tube (CRT) to display information. The monitor is associated with a keyboard for manual input of characters and displays the information as it is keyed in. It also displays the program or application output. Like the television, monitors are also available in different sizes. • **Printer**: Printers are used to produce paper (commonly known as hard copy) output. Based on the technology used, they can be classified as Impact or Non-impact printers.

Impact printers use the typewriting printing mechanism wherein a hammer strikes the paper through a ribbon in order to produce output. Dot-matrix and Character printers fall under this category.

Non-impact printers do not touch the paper while printing. They use chemical, heat or electrical signals to etch the symbols on paper. Inkjet, Deskjet, Laser, Thermal printers fall under this category of printers.

Plotter: Plotters are used to print graphical output on paper. It interprets computer commands and makes line drawings on paper using multi colored automated pens. It is capable of producing graphs, drawings, charts, maps etc. • **Facsimile (FAX)**: Facsimile machine, a device that can send or receive pictures and text over a telephone line. Fax machines work by digitizing an image.

Sound cards and Speaker(s): An expansion board that enables a computer to manipulate and output sounds. Sound cards are necessary for nearly all CD-ROMs and have become commonplace on modern personal computers. Sound cards enable the computer to output sound through speakers connected to the board, to record sound input from a microphone connected to the computer, and manipulate sound stored on a disk.

WORKING WITH NUMBERS

MATLAB (**mat**rix **lab**oratory) is a multi-paradigm numerical computing environment and fourth-generation programming language. A proprietary programming language developed by MathWorks, MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages, including C, C++, C#, Java, Fortran and Python.

Although MATLAB is intended primarily for numerical computing, an optional toolbox uses the MuPAD symbolic engine, allowing access to symbolic computing abilities. An additional package, Simulink, adds graphical multi-domain simulation and model-based design for dynamic and embedded systems.

As of 2017, MATLAB has over 2 million users across industry and academia. MATLAB users come from various backgrounds of engineering, science, and economics.

MACHINE CODE

- Machine Code
- MATLAB programming language
- Military GPS signal (or GPS_signals#Military_.28M-code.29), or half of the **G & M**-Code programming language used in the CNC Machining Industry.

Every processor or processor family has its own machine code instruction set. Instructions are patterns of bits that by physical design correspond to different commands to the machine. Thus, the instruction set is specific to a class of processors using (mostly) the same architecture.

Successor or derivative processor designs often include all the instructions of a predecessor and may add additional instructions.

Occasionally, a successor design will discontinue or alter the meaning of some instruction code (typically because it is needed for new purposes), affecting code compatibility to some extent; even nearly completely compatible processors may show slightly different behavior for some instructions, but this is rarely a problem. Systems may also differ in other details, such as memory arrangement, operating systems, or peripheral devices. Because a program normally relies on such factors, different systems will typically not run the same machine code, even when the same type of processor is used.

A machine code instruction set may have all instructions of the same length, or it may have variable-length instructions. How the patterns are organized varies strongly with the particular architecture and often also with the type of instruction. Most instructions have one or more opcode fields which specifies the basic instruction type (such as arithmetic, logical, jump, etc.) and the actual operation (such as add or compare) and other fields that may give the type of the operand(s), the addressing mode(s), the addressing offset(s) or index, or the actual value itself (such constant operands contained in an instruction are called *immediates*).[2]

Not all machines or individual instructions have explicit operands. An accumulator machine has a combined left operand and result in an implicit accumulator for most arithmetic instructions. Other architectures (such as 8086 and the x86-family) have accumulator versions of common instructions, with the accumulator regarded as one of the general registers by longer instructions. A stack machine has most or all of its operands on an implicit stack. Special purpose instructions also often lack explicit operands (CPUID in the x86 architecture writes values into four implicit destination registers, for instance). This distinction between explicit and implicit operands is important in machine code generators, especially in the register allocation and live range tracking parts. A good code optimizer can track implicit as well as explicit operands which may allow more frequent constant propagation, constant folding of registers (a register assigned the result of a constant expression freed up by replacing it by that constant) and other code enhancements.

Programs

A computer program is a sequence of instructions that are executed by a CPU. While simple processors execute instructions one after another, superscalar processors are capable of executing several instructions at once.

Program flow may be influenced by special 'jump' instructions that transfer execution to an instruction other than the numerically following one. Conditional jumps are taken (execution

continues at another address) or not (execution continues at the next instruction) depending on some condition.

Assembly languages

A much more readable rendition of machine language, called assembly language, uses mnemonic codes to refer to machine code instructions, rather than using the instructions' numeric values directly. For example, on the Zilog Z80 processor, the machine code `00000101`, which causes the CPU to decrement the `B` processor register, would be represented in assembly language as `DEC B`.

Example

The MIPS architecture provides a specific example for a machine code whose instructions are always 32 bits long. The general type of instruction is given by the *op* (operation) field, the highest 6 bits. J-type (jump) and I-type (immediate) instructions are fully specified by *op*. R-type (register) instructions include an additional field *funct* to determine the exact operation. The fields used in these types are:

```
  6     5     5     5     5     6 bits
[ op  |  rs |  rt |  rd |shamt| funct]  R-type
[ op  |  rs |  rt | address/immediate]  I-type
[ op  |       target address       ]  J-type
```

rs, *rt*, and *rd* indicate register operands; *shamt* gives a shift amount; and the *address* or *immediate* fields contain an operand directly.

For example, adding the registers 1 and 2 and placing the result in register 6 is encoded:

```
[ op  |  rs |  rt |  rd |shamt| funct]
   0     1     2     6     0     32     decimal
 000000 00001 00010 00110 00000 100000   binary
```

Load a value into register 8, taken from the memory cell 68 cells after the location listed in register 3:

```
[ op  |  rs |  rt | address/immediate]
  35     3     8          68          decimal
 100011 00011 01000 00000 00001 000100   binary
```

Jumping to the address 1024:

```
[ op |       target address       ]
   2              1024             decimal
 000010 00000 00000 00000 10000 000000   binary
```

Relationship to microcode

In some computer architectures, the machine code is implemented by an even more fundamental underlying layer called microcode, providing a common machine language interface across a line or family of different models of computer with widely different underlying dataflows. This is done to facilitate porting of machine language programs between different models. An example of this use is the IBM System/360 family of computers and their successors. With dataflow path widths of 8 bits to 64 bits and beyond, they nevertheless present a common architecture at the machine language level across the entire line.

Using microcode to implement an emulator enables the computer to present the architecture of an entirely different computer. The System/360 line used this to allow porting programs from earlier IBM machines to the new family of computers, e.g. an IBM 1401/1440/1460 emulator on the IBM S/360 model 40.

Relationship to byte code

Machine code is generally different than byte code (also known as p-code), which is either executed by an interpreter or itself compiled into machine code for faster (direct) execution. An exception is when a processor is designed to use a particular byte code directly as its machine code, such as is the case with Java processors.

Machine code and assembly code are sometimes called *native code* when referring to platform-dependent parts of language features or libraries.

Storing in memory

The Harvard architecture is a computer architecture with physically separate storage and signal pathways for the code (instructions) and data. Today, most processors implement such separate signal pathways for performance reasons but actually implement a Modified Harvard architecture,[*citation needed*] so they can support tasks like loading an executable program from disk storage as data and then executing it. Harvard architecture is contrasted to the Von Neumann architecture, where data and code are stored in the same memory which is read by the processor allowing the computer to execute commands.

From the point of view of a process, the *code space* is the part of its address space where the code in execution is stored. In multitasking systems this comprises the program's code segment and usually shared libraries. In multi-threading environment, different threads of one process share code space along with data space, which reduces the overhead of context switching considerably as compared to process switching.

Readability by humans

It has been said that machine code is so unreadable that the United States Copyright Office cannot identify whether a particular encoded program is an original work of authorship; however, the US Copyright Office *does* allow for copyright registration of computer programs[5] and a program's machine code can sometimes be decompiled in order to make its functioning more easily understandable to humans.

Cognitive science professor Douglas Hofstadter has compared machine code to genetic code, saying that "Looking at a program written in machine language is vaguely comparable to looking at a DNA molecule atom by atom.

SOFTWARE HIERARCHY

The lowest level description of a computer program is just the sequence of numbers which encode the basic CPU operations. This level is called **machine code**. Machine code is specific to a given CPU manufacturer and often specific to a given model type (for example the Pentium CPU has some codes not used by earlier 8086 CPUs). Machine code is very difficult for a human to read or write, so the lowest level of programming done by humans is in a language in which each basic operation is given a mnemonic code called **assembly language**. Humans can read and write using assembly language which can be converted into machine code using an **assembler**. Assembly language, like machine code is often specific to a particular CPU manufacturer or model.

The development of **high-level languages** meant that humans could program using a formalism that was closer to their conceptual models of the data being manipulated: characters, real numbers, lists, tables or database records. Such languages are easier for humans to learn and to use, and furthermore they tend to be available across different computers; with each manufacturer supplying a conversion program between the high-level language and the assembly language for their CPU. Examples of high-level languages are Fortran, Pascal, Basic, C, C++, Java and MATLAB.

Modern computer systems need to deal with complex tasks involving multiple programs interacting simultaneously, and the sharing of access to files on disks, to network resources and displays. To cope with these demands, manufacturers supply **operating systems** (e.g. Windows, Linux), which are themselves programs which help the user operate the computer and run

other **application** programs. Often individual application programs need to work together to achieve an objective: for example a word processing application might call on a drawing package or on a spreadsheet program to do some specific processing within a document. This idea of combining programs is called **scripting**, where the specifications for which programs are to be executed and how they should interact is specified in a **script**.

PART-B(2 MARKS)

POSSIBLE QUESTIONS

1. What is Machine Code?
2. What is Software hierarchy**?**
3. Mention any four components of a computer.

PART-C(6 MARKS)

POSSIBLE QUESTIONS

1. Explain Components of a computer in detail.
2. Describe about Working with numbers and Machine code
3. Discuss about Machine code and Software hierarchy

KARPAGAM ACADEMY OF HIGHER EDUCATION
Department of Computer Science
II B.Sc(CS) (BATCH 2016-2018)
Programming In MATLAB
PART-A OBJECTIVE TYPE/ MULTIPLE CHOICE QUESTIONS

	ONLINE EXAMINATIONS **ONE MARK QUESTI**
S.No	**Question**
1	Which is not a computer classification?
2	A computer program that converts an entire program into machine language is called _____
3	RAM is a _____ memory.
4	RAM stands for _____ _____Memory.
5	Hard Disk is an example for _____ memory
6	Computer system comprises of major units
7	A computer program that converts an entire program into machine language line by line is called _____.
8	Intel corporation produces _____
9	_____ do billion calculations in one second.
10	Central Processing Unit is combination of
11	Mouse is an example for _____ device.
12	Which unit converts user data into machine readable form?
13	_____ is an example for output device.
14	_____ is an example for output device,
15	Through which device the main components of the computer communicate with each other?
16	What type of device is plotter?
17	Vacuum Tubes were replaced by _____.
18	_____ is faster than inkjet printer.
19	Software is a set of _____.
20	_____ software is useful for specific application.
21	Which of the following produces the best quality graphics reproduction?
22	OCR stands _____ _____ Reader.
23	_____ is an example for Printer.
24	CD stands for _____ _____.
25	DVD is an example for _____.
26	____ converts the programs written in assembly language into machine instructions .
27	The instructions like MOV or ADD are called as _____ .

28	Instructions which wont appear in the object program are called as _____ .
29	The assembler stores all the names and their corresponding values in _____ .
30	The assembler stores the object code in _____ .
31	The register used to store the flags is called as _____ .
32	_____ is an example for output device.
33	_____ is an example for output device,
34	What is the responsibility of the logical unit in the CPU of a computer?
35	*Punch Card System* was developed by _____.
36	The computer size was very large in
37	The section of the CPU that is responsible for performing mathematical operations
38	The brain of any computer system is _____.
39	Primary memory stores
40	The word length of a computer is measured in
41	LSI stands for Large _____ _____.
42	What is the responsibility of the logical unit in the CPU of a computer?
43	_____is an input device that utilizes a light-sensitive detector to select objects on a display screen.
44	A is a device for pointing (controlling input positioning) on a computer display screen
45	A _____ is an input device used to enter motion data into computers or other electronic devices
46	_____ are photoelectric scanners that read the bar codes
47	_____ is specific to a given CPU manufacturer and often specific to a given model type
48	This idea of combining programs is called ____ where the specifications for which programs are to be executed and how they should interact is specified in a script.
49	_____ function is used to find the minimum of given numbers
50	_____ function is used to find the maximum of given numbers

ONS		
Option 1	**Option 2**	**Option 3**
mainframe	min	maxframe
Interpreter	Assembler	Compiler
Rigid Access	Right Access	Rom Access
Read Access	Random Access	Rough Access
Secondary	Primary	Tertiary
input unit, output unit, control unit	input unit, output unit, control unit and storage	input unit, output unit, central processing unit and storage unit
Interpreter	Simulator	Compiler
Microprocessor	CD	DVD
Mainframe Computers	Mini Computer	Micro computer
Control and storage	Control and output unit	Arithmetic logic and input unit
Input	Output	Programming
input	Output unit	ALU
Pendrive	Monitor	Memory unit
printer	keyboard	mircroprocessor
Keyboard	System Bus	Monitor
Memory	Output	Storage
Transistors	memory chips	valves
Laser Printer	Dot Matrix	Radar Printer
compuer disks	computer chips	computer programs
Application	Simulator	Emulator
Laser Printer	Inkjet Printer	Plotter
Optional Character	Operation Character	Optical Character
Daisy Wheel	Dolby Wheel	David Wheel
Cobined Disk	Cumulative Disk	Cop Disk
RAM	ROM	VRAM
Machine compiler	Interpreter	Assembler
OP-Code	Operators	Commands

Redundant instructions	Exceptions	Comments
Special purpose Register	Symbol Table	Value map Set
Main memory	Cache	RAM
Flag register	Status register	Test register
Pendrive	Monitor	Memory unit
printer	keyboard	mircroprocessor
To produce result	To compare numbers	To control flow of information
Jacquard	John	Jogo Napier
First Generation	Second Generation	Third Generation
Memory	Register Unit	Control Unit
ALU	CPU	Memory unit
Data alone	Programs alone	Results alone
Bytes	Millimeters	Meters
Scale Integration	Slot Integration	Slow Integration
To produce result	To compare numbers	To control flow of information
touch pad	Track ball	keyboard
Barcode	keyboard	Track ball
Track ball	touch pad	Barcode
Bar-code readers	keyboard	Track ball
scripting	Software hierarchy	Machine code
Software hierarchy	Machine code	Components
min	max	medium
poor	min	max

Option 4			**Answers**
notebook			maxframe
Commander			Compiler
Random Acesss			Random Access
Right Access			Random Access
Territory			Secondary
input, output and storage units			input unit, output unit, central processing unit and storage unit
Commander			Interpreter
PEN DRIVE			Microprocessor
Super computer			Super computer
Arithmetic logic and control unit			Arithmetic logic and control unit
Printing			Input
Control unit			input
Registers			Monitor
mouse			printer
Memory			System Bus
input			Output
capacitor			Transistors
Tape Printer			Laser Printer
computer memory			computer programs
Desktop			Application
Dot matrix printer			Plotter
Oppo Character			Optical Character
Darwin Wheel			Daisy Wheel
Compact Disk			Compact Disk
DRAM			ROM
Converter			Assembler
None			OP-Code

Assembler Directives			Assembler Directives
None			Symbol Table
Magnetic disk			Magnetic disk
Log register			Status register
Registers			Monitor
mouse			printer
To do math’s works			To compare numbers
Jackson			Jacquard
Fourth Generation			First Generation
ALU			ALU
Control unit			CPU
All the above			All the above
Bits			Bits
Sum Integration			Scale Integration
To do math’s works			To compare numbers
Barcode			Light pen
touch pad			touch pad
keyboard			trackball
touch pad			Bar-code readers
Components			Machine code
scripting			scripting
poor			min
medium			max

UNIT II

SYLLABUS

Programming Environment: MATLAB Windows, A First Program, Expressions, Constants, Variables and assignment statement, Arrays.

Programming Environment

MATLAB WINDOWS

It is assumed that the software is installed on the computer, and that the user can start the program. Once the program starts, the MATLAB desktop window opens (Figure 1-1). The window contains four smaller windows: the Command Window, the Current Folder Window, the Workspace Window, and the Command History Window. This is the default view that shows four of the various windows of MATLAB. A list of several windows and their purpose is given in Table 1-1. The Start button on the lower left side can be used to access MATLAB tools and features. Four of the windows—the Command Window, the Figure Window, the Editor Window, and the Help Window—are used extensively throughout the book and are briefly described on the following pages

Command Window: The Command Window is MATLAB's main window and opens when MATLAB is started. It is convenient to have the Command Window as the only visible window, and this can be done by either closing all the other windows (click on the x at the top right-hand side of the window you want to close)

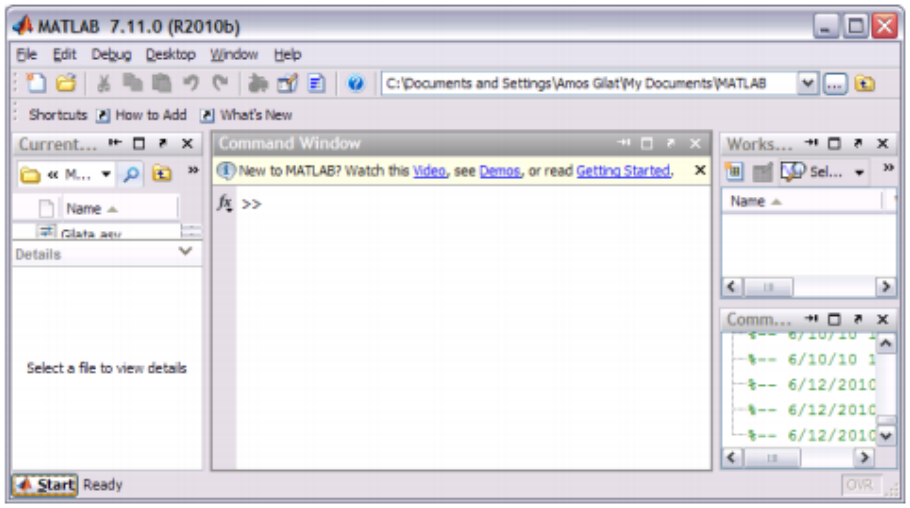


Figure 1-1: The default view of MATLAB desktop.

Table 1-1: MATLAB windows

Window	Purpose
Command Window	Main window, enters variables, runs programs.
Figure Window	Contains output from graphic commands.
Editor Window	Creates and debugs script and function files.
Help Window	Provides help information.
Command History Window	Logs commands entered in the Command Window.
Workspace Window	Provides information about the variables that are used.
Current Folder Window	Shows the files in the current folder.

Figure Window: The Figure Window opens automatically when graphics commands are executed, and contains graphs created by these commands. An example of a Figure Window is shown in Figure 1-2.

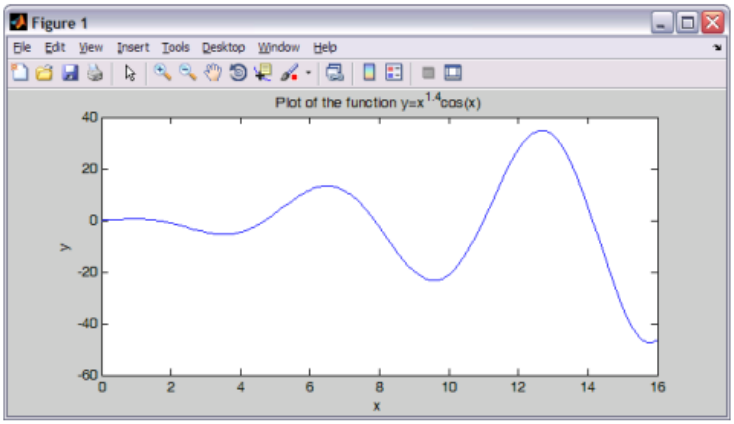


Figure 1-2: Example of a Figure Window.

Editor Window: The Editor Window is used for writing and editing programs. This window is opened from the File menu. An example of an Editor Window is shown in Figure 1-3.

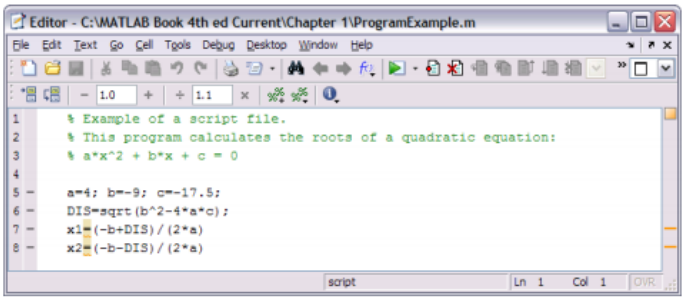


Figure 1-3: Example of an Editor Window.

Help Window: The Help Window contains help information. This window can be opened from the Help menu in the toolbar of any MATLAB window. The Help Window is interactive and can be used to obtain information on any feature of MATLAB. Figure 1-4 shows an open Help Window.

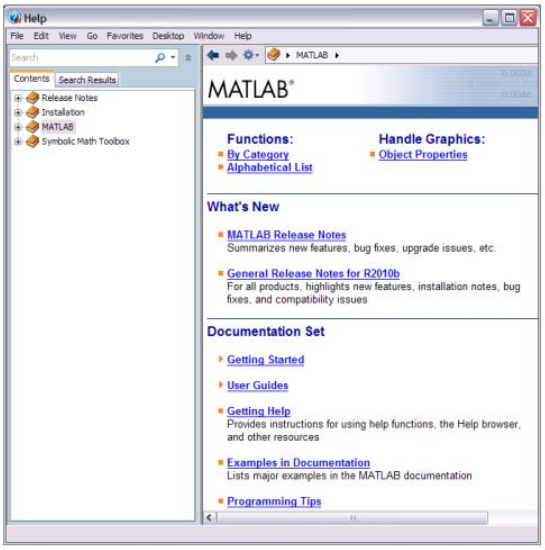


Figure 1-4: The Help Window.

Working In The Command Window The Command Window is MATLAB's main window and can be used for executing commands, opening other windows, running programs written by the user, and managing the software. An example of the Command Window, with several simple commands that will be explained later in this chapter, is shown in Figure 1-5.

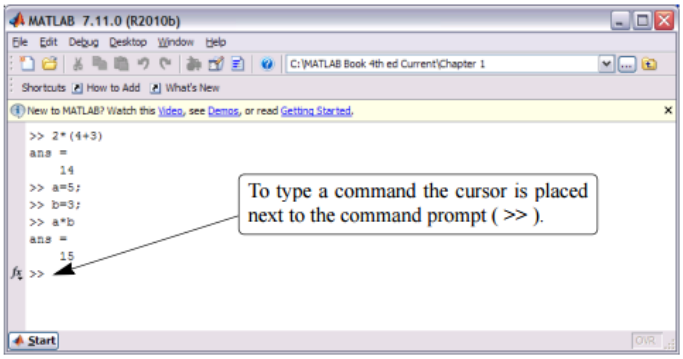


Figure 1-5: The Command Window.

A FIRST PROGRAM

Matlab stores most of its numerical results as matrices. Unlike some languages (C, C++, C#), it dynamically allocates memory to store variables. Therefore, it is not necessary to declare variables before using them. Let's begin by simply adding two numbers. Click in the Command Window. You will see a flashing "|" symbols next to the ">>" symbol. Enter the following commands

1. Type in "x = 3" then hit "enter"
2. Type in "y = 2;" then hit "enter" (note the semicolon here!)
3. Type "z = x + y" then hit "enter"

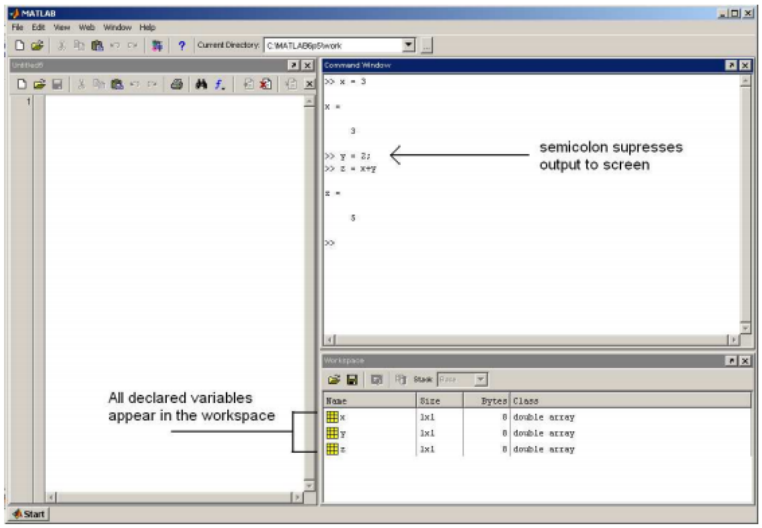


Figure 4: Entering in scalar values into **Matlab**

All declared variables appear in the workspace. Recall that these values are stored as matrices. The "size" column tells us the dimension of the matrix. As expected, all these variables are 1x1 scalar values. To double check on value stored in this matrix, simply double click any of the variables in the Workspace.

Example program

The command

disp(*argument*);

displays the value of the argument. This can be a number, a string in single quotes, or an expression. For simple numbers, the arithmetic operators are: **+, -, *, /** and**^.** Try

disp(2*3+1);

or

disp('Hello World!');

Try these programs out first on the command line; then practise using the editor to enter the commands, saving them to a file, loading the file and running the program from inside the editor.

Expressions

VARIABLES

Like most other programming languages, the MATLAB® language provides mathematical *expressions*, but unlike most programming languages, these expressions involve entire matrices.

MATLAB does not require any type declarations or dimension statements. When MATLAB encounters a new variable name, it automatically creates the variable and allocates the appropriate amount of storage. If the variable already exists, MATLAB changes its contents and, if necessary, allocates new storage. For example,

```
num_students = 25
```

creates a 1-by-1 matrix named num_students and stores the value 25 in its single element. To view the matrix assigned to any variable, simply enter the variable name.

Variable names consist of a letter, followed by any number of letters, digits, or underscores. MATLAB is case sensitive; it distinguishes between uppercase and lowercase letters. A and a are *not* the same variable.

Although variable names can be of any length, MATLAB uses only the first N characters of the name, (where N is the number returned by the function namelengthmax), and ignores the rest. Hence, it is important to make each variable name unique in the first N characters to enable MATLAB to distinguish variables.

```
N = namelengthmax

N =     63
```

Numbers

MATLAB uses conventional decimal notation, with an optional decimal point and leading plus or minus sign, for numbers. Scientific notation uses the letter e to specify a power-of-ten scale factor. Imaginary numbers use either i or j as a suffix. Some examples of legal numbers are

```
3           -99          0.0001

9.6397238   1.60210e-20  6.02252e23

1i          -3.14159j    3e5i
```

MATLAB stores all numbers internally using the *long* format specified by the IEEE® floating-point standard. Floating-point numbers have a finite *precision* of roughly 16 significant decimal digits and a finite *range* of roughly 10^{-308} to 10^{+308}.

Numbers represented in the double format have a maximum precision of 52 bits. Any double requiring more bits than 52 loses some precision. For example, the following code shows two unequal values to be equal because they are both truncated:

```
x = 36028797018963968;

y = 36028797018963972;

x == y

ans =

     1
```

Integers have available precisions of 8-bit, 16-bit, 32-bit, and 64-bit. Storing the same numbers as 64-bit integers preserves precision:

```
x = uint64(36028797018963968);
y = uint64(36028797018963972);
x == y
ans = 0
```

Matrix Operators

Expressions use familiar arithmetic operators and precedence rules.

+	Addition
-	Subtraction
*	Multiplication
/	Division
\	Left division
^	Power
'	Complex conjugate transpose
( )	Specify evaluation order

Array Operators

When they are taken away from the world of linear algebra, matrices become two-dimensional numeric arrays. Arithmetic operations on arrays are done element by element. This means that addition and subtraction are the same for arrays and matrices, but that multiplicative operations are different. MATLAB uses a dot, or decimal point, as part of the notation for multiplicative array operations.

The list of operators includes

+	Addition
-	Subtraction
.*	Element-by-element multiplication
./	Element-by-element division
.\	Element-by-element left division
.^	Element-by-element power
.'	Unconjugated array transpose

If the Dürer magic square is multiplied by itself with array multiplication

```
A.*A
```

the result is an array containing the squares of the integers from 1 to 16, in an unusual order:

```
ans =
   256     9     4   169
    25   100   121    64
```

```
    81    36    49   144
    16   225   196     1
```

Examples of Expressions

You have already seen several examples of MATLAB expressions. Here are a few more examples, and the resulting values:

```
rho = (1+sqrt(5))/2
rho =
   1.6180


a = abs(3+4i)
a =
     5


z = sqrt(besselk(4/3,rho-i))
z =
  0.3730+ 0.3214i


huge = exp(log(realmax))
huge =
  1.7977e+308


toobig = pi*huge
toobig =
 Inf
```

VARIABLES AND ASSIGNMENT

Variables are named locations in memory where numbers, strings and other elements of data may be stored while the program is working. Variable names are combinations of letters and digits, but must start with a latter. MATLAB does not require you to declare the names of variables in advance of their use. This is actually a common cause of error, since it allows you to

refer accidentally to variables that don't exist. To assign a variable a value, use the **assignment statement**. This takes the form

variable=expression;

for example

a=6;

or

name='Mark';

To display the contents of a variable, use

disp(*variable*);

Please note that −

- Once a variable is entered into the system, you can refer to it later.
- Variables must have values before they are used.
- When an expression returns a result that is not assigned to any variable, the system assigns it to a variable named ans, which can be used later.

For example,

```
sqrt(78)
```

MATLAB will execute the above statement and return the following result −

```
ans =  8.8318
```

You can use this variable **ans** −

```
sqrt(78);
9876/ans
```

MATLAB will execute the above statement and return the following result −

```
ans =  1118.2
```

Let's look at another example −

```
x = 7 * 8;
y = x * 7.89
```

MATLAB will execute the above statement and return the following result −

```
y =  441.84
```

Multiple Assignments

You can have multiple assignments on the same line. For example,

```
a = 2; b = 7; c = a * b
```

MATLAB will execute the above statement and return the following result −

```
c = 14
```

ARRAYS

MATLAB is particularly powerful in the way it deals with tables of data, called arrays. An array is simply a variable that can contain a number of values arranged in tabular form. Arrays may be one dimensional (like a list), two dimensional (like a table), or have more dimensions. To set the value of one element of a one dimensional array, use the notation

variable(*index*)=*expression*;

for example

```
table(1)=3;
table(2)=6;
```

Note that indexes must be expressions evaluating to positive integers. The smallest index is 1. To access one element from a one dimensional array, use the notation

variable(*index*)

for example

```
a=table(2);
disp(table(2));
```

For two dimensional arrays, use

variable(*index*,*index*)=*expression*;
to set the value and

variable(*index*,*index*)
to retrieve its value. You can store strings in tables, but each string occupies a row, and all rows must be the same length (think of a two-dimensional array of characters).

You can assign a whole array in one operation using a notation involving square brackets: for example:

array = [v11 v12 v13; v21 v22 v23];
where v11 is the value in row 1 col 1; v21 is the value in row 2 col 1; etc. The ';' marks the end of a row.

You can generate arrays containing sequences very easily with the ':' operator. The expression

start:*stop*
generates a sequence of integers from start to stop. The expression

start:*increment*:*stop*
generates a sequence from start to stop with the specfied increment. Try

```
disp(1:10);
disp(1:2:10);
```
You can also select sub-parts of the array with the ':' operator. For example,

x(3:5)
represents the array consisting of the third through fifth elements of x. Also

y(2:2:100)
represents the array containing the even number elements of y below index 100.

You can also add subtract, multiply and divide arrays of data using the operators we've mentioned previously. However MATLAB makes a difference between operations that work on a cell-by-cell basis (so-called "dot" operations) as opposed to operations that work on the arrays as a whole. For example, if you want to multiply two arrays of equal size to give a third array in which each cell contains the product of the corresponding cells in the input, then you need to use the "dot-multiply" operator .* for example

```
C = A.*B;
```
Finally you can transpose rows and columns of a matrix with the ' operator, for example

```
disp(A')
```

PART-B(2 MARKS)

POSSIBLE QUESTIONS

1. Explain what is MATLAB? Where MATLAB can be applicable?
2. List out the operators that MATLAB allows?
3. What does MATLAB consist of?
4. What is a variable in MATLAB?
5. What is an Expression? Give one example.
6. What is an Array**?**
7. What is Constant?

PART-C(6 MARKS)

POSSIBLE QUESTIONS

1. Explain in detail about MATLAB Windows with neat sketch.
2. Write a note on Expressions, Constants and Variables.
3. Explain in detail about array and its types with suitable example.

KARPAGAM ACADEMY OF HIGHER EDUCATION
Department of Computer Science
II B.Sc(CS) (BATCH 2016-2018)
Programming In MATLAB
PART-A OBJECTIVE TYPE/ MULTIPLE CHOICE QUESTIC

	ONLINE EXAMINATIONS	**ONE MAR**
S.No	**Question**	**Option 1**
1	MATLAB stands for _____	Maths Laboratory
2	MATLAB was developed by _____	MathsWorks
3	In MATLAB the matrix is defined as an _____	vector
4	_____ acts as an outstanding tool for visulaizing technical data	C
5	The fundamental unit od data in any MATLAB program is the _____	array
6	In command window the _____ are entered	datas
7	_____ window displays plots and graphs	command
8	_____ window permits a user to create and modify MATLAB programs	command
9	MATLAB programs are saved with the extensions _____	.m
10	The _____ window displays a list of commands that a user has entered in the command window	edit
11	When a window is _____ it appears as a pane within the MATLAB desktop	docked
12	A _____ is a collection of all the variables and arrays that can be used by MATLAB when a particular command is executed	editor
13	_____ command is used to list all the variables and arrays in the current workspace	string
14	A variable can be deleted from the workspace with the _____ command	delete
15	The _____ command will display a list of possible help topics in the command window	help
16	The _____ command searches the quick summary information in each function for a match	lookfor
17	The term _____ is used to describe an array with only one dimension	array
18	The term _____ is used to describe an array with two or more dimensions	array
19	A variable o ftype _____ is automatically created whenever a numerical value is assigned to a variable name	double
20	MATLAB is a _____ typed language	strongly
21	The _____ operator swaps the row and columns of any array that is given	transpose
22	The _____ function can be used ti create an all zero array	ones

23	The _____ function can be used to generate arrays containing all ones	ones
24	The eye function can be used to generate arrays coantining _____ matrices	square
25	MATLAB always allocated array elements in _____ major order	row
26	The _____ functions returns the highest value taken on by that subscript	ones
27	pi is an example of _____	operators
28	NaN stands for _____	Number and number
29	The default format for displaying the output can be changed by using _____ command	path
30	The _____ function accepts an array arugment and displayus the value of athe array in the command window	disp
31	the _____ function displays one or more values together with realted text	disp
32	The _____ command loads data from a disk file into the current MATLAB WORKSPACE	save
33	_____ are operations performed between arrays on an element by element basis	matrix operations
34	In _____ the number of rows and columns in both arrays must be the same	matrix operations
35	The MATLAB functions can return _____ results to the calling program	more than one
36	The _____ command can be used to save a plot as a graphical image by specifying appropriate options and a filename	plot
37	If the result of the MATLAB expression is not assigned to any variable, then itr is stored in default variable _____	result
38	The _____ gives the transpose of x	x'
39	What symbol precedes all comments in MAtlab?	''
40	Which of the following is not pre defines variable in Matlab	pi
41	this matlab command clears all data and variables stored in memory	clc
42	characters in matlab are represented in their values in memory	decimal
43	A correct name for a variable is	1arearec
44	An incorrect name for a variable is	cat1
45	The _____ function converts numerical data to logical data	real
46	the _____ fucntion converts logical data to numerical data	real
47	The _____ operators are operators with two numerical or string operands that yield a logical result	logical
48	The relational operators can compare two strings only if they are of _____ length	equal
49	the operator, == stands for	not equal to
50	To join two or more statements with an or condition use the operator	&

ONS

K QUESTIONS					
Option 2	**Option 3**	**Option 4**			**Answers**
Matrix Laboratory	Mathematical Lab	Maths Lab			Matrix Laboratory
Intel	Microsoft	IBM			MathsWorks
scalar	array	integer			array
C++	Java	MATLAB			MATLAB
vector	scalar	none			array
values	commands	fiels			commands
Edit	Figure	Command history			Figure
Edit	Figure	Command history			Edit
.mm	.mf	.ml			.m
figure	debug	command history			command history
undocked	removed	deleted			docked
workspace	desktop	none			workspace
whos	whose	where			whos
remove	clear	omit			clear
helper	lookfor	order			help
helper	help	order			lookfor
vector	matrix	scale			vector
vector	matrix	scale			matrix
long	int	short			double
weakly	stronger	thiner			weakly
concatenates	colon	semicolon			transpose
zero	eye	randn			zero

zero	eye	randn			ones
null	identity	none			identity
column	row & column	none			column
zero	end	repalce			end
functions	plotting	special values			special values
number	not a number	not and number			not a number
format	special	null			format
format	special	fprintf			disp
format	fprintf	special			fprintf
update	load	open			load
array operations	vector operations	arthimetic operations			array operations
array operations	vector operations	arthimetic operations			array operations
exactly one	only two	none			more than one
print	draw	multiple			print
ans	answer	output			ans
x''	x'''	x			x'
%	//	none			none
inf	i	gravity			gravity
clear	delete	deallocate			clear
ASCII	hex	string			ASCII
area rec	area_rec	cos			area_rec
cat_1	cat_cos	1cat			1cat
logical	relation	array			logical
logical	relation	array			real
relational	bitwise	arithmetic			relational
different	both a&b	none			equal
equal to	assigned to	approximately equal to			equal to
|	or	U			|

UNIT-III

SYLLABUS

Graph Plots: Basic plotting, Built in functions, Generating waveforms, Sound replay, load and save. Procedures and Functions: Arguments and return values, M-files, Formatted console input-output , String handling

GRAPH PLOTS

BASIC PLOTTING:

To create XY graphs, it is easiest to form your data into two row vectors, one for the x co-ordinates, and one for the y co-ordinates. The command

```
plot(x,y)
```

will then create a figure with points at each y value for each matching x value. You can control the style of any line drawn through the points by a third string argument to the plot command:

```
plot(x,y,style);
```

where style is made up from characters as follows:

- Color strings are 'c', 'm', 'y', 'r', 'g', 'b', 'w', and 'k'. These correspond to cyan, magenta, yellow, red, green, blue, white, and black.
- Linestyle strings are '-' for solid, '--' for dashed, ':' for dotted, '-.' for dash-dot, and none for no line.

The marker types are '+', 'o', '*', and 'x' and the filled marker types 's' for square, 'd' for diamond, '^' for up triangle, 'v' for down triangle, '>' for right triangle, '<' for left triangle, 'p' for pentagram, 'h' for hexagram, and none for no marker.

For example:

```
x = [ 1 2 3 4 ];
y = [ 10 15 20 25 ];
plot(x,y,'g-*');
```

You can plot multiple lines by repeating the arguments:

```
plot(x1,y1,x2,y2,…);
```

or

```
plot(x1,y1,style1,x2,y2,style2,…);
```

You can give the graph a title with the

title(*label*);

command, where label is a character string. Likewise you can add labels to the X and Y axes with

xlabel(*label*);

and

ylabel(*label*);

You can add a legend with

legend(*label1*,*label2*,*label3*,…);

Description

Plotting functions accept line specifications as arguments and modify the graph generated accordingly. You can specify these three components:

- Line style
- Marker symbol
- Color

Line Style Specifiers

You indicate the line styles, marker types, and colors you want to display, detailed in the following tables:

Specifier	LineStyle
'-'	Solid line (default)
'--'	Dashed line
':'	Dotted line
'-.'	Dash-dot line

Marker Specifiers

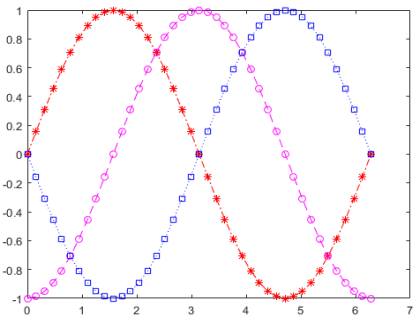
Specifier	Marker Type
'+'	Plus sign
'o'	Circle

Specifier	Marker Type
'*'	Asterisk
'.'	Point
'x'	Cross
'square' or 's'	Square
'diamond' or 'd'	Diamond
'^'	Upward-pointing triangle
'v'	Downward-pointing triangle
'>'	Right-pointing triangle
'<'	Left-pointing triangle
'pentagram' or 'p'	Five-pointed star (pentagram)
'hexagram' or 'h'	Six-pointed star (hexagram)

Color Specifier

Specifier	Color
r	Red
g	Green
b	Blue
c	Cyan
m	Magenta
y	Yellow
k	Black
w	White

```
t = 0:pi/20:2*pi;
plot(t,sin(t),'-.r*')
hold on
plot(t,sin(t-pi/2),'--mo')
plot(t,sin(t-pi),':bs')
hold off
```



BUILT IN FUNCTIONS

Generation

zeros()	matrix of specified size filled with zeros
ones()	matrix of specified size filled with ones
rand()	generate pseudo random number(s) between 0 and 1

Arithmetic

rem()	remainder after integer division
abs()	absolute value (also character -> number)
fix()	truncate a value to its integer part (towards zero)
round()	round a value to nearest integer.
sqrt()	square root
sin()	sine (angle in radians)
cos()	cosine (angle in radians)
exp()	exponential
log()	natural logarithm
log10()	logarithm base 10

Status

length()	length of a vector (longest dimension of matrix)
size()	size of a matrix [nrows, ncols]

Miscellaneous

sum()	sum the elements of a vector
mean()	find mean of elements of a vector
sort()	sort the elements of a vector in increasing size
clock()	returns date and time as a vector [year month day hour minute seconds]
date()	returns date as a string dd-mmm-yyyy

GENERATING WAVEFORMS

Waveforms are just long vectors with one number per amplitude sample. Usually they are best kept scaled so that each amplitude is between –1 and 1. To generate a sinewave, first generate a time sequence t representing the times of each sampling instant; for example:

```
t = 0:0.0001:2;
```

would generate a two second sequence with a sampling interval of 0.1ms (i.e. 10,000Hz). You can then generate a sinewave at frequency F with the expression

```
y = sin(2*pi*F*t);
```

You can create a pulse by creating a vector of zeros and setting a single element to one. A pulse train has a series of elements set to one. If these occurred every 100 elements, you might use the expression

```
y(1:100:10000)=1;
```

To create a simple sawtooth, you can use the remainder function, for example

```
y = rem(1:10000,100)/100;
```

To create a noise waveform, you can use the 'rand(*nrows*,*ncols*)' function, for example

```
y = rand(1,10000);
```

SOUND REPLAY, LOAD AND SAVE

To replay a waveform, you can use

sound(*wave*,*samplerate*);

To ensure that the waveform is scaled to the range –1 .. +1 before replay, use

soundsc(*wave*,*samplerate*);

instead.

To save a waveform to a file, use

save *filename variable*;

To load a waveform from a file, use

load *filename variable*;

To save a waveform in a Windows compatible audio file format, use

wavwrite(*waveform*,*samplerate*,*filename*);

To load a Windows compatible audio file, use

[*waveform*,*samplerate*,*nbits*]=wavread(*filename*);

PROCEDURE AND FUNCTIONS: ARGUMENTS AND RETURN VALUES

Functions

You can define your own functions to complement those provided by MATLAB. Functions are the building blocks of your own programs. You should always try and divide your programming task into separate functions, then design, code and test each one independently. It is common to design from the top down, but build from the bottom up.

It is good practice to store each function in its own source file, with the name of the source file matching the function. Thus a function called "myfunc" will be stored in the file "myfunc.m". This way, both you and MATLAB can easily find the source file for a function given its name. The first line of a function source file should then be the function definition line, which has the format:

function *outargs*=*funcname*(*inargs*);

The function name can be a mixture of letters and digits but must start with a letter. It is a good idea to avoid names that MATLAB is already using. The *inargs*parameter is a list of variable names separated by commas. These are the dummy names you will use in the code for the function to 'stand for' the actual arguments passed to the function when it is executed. Likewise the *outargs* parameter is a list of variable names separated by commas which stand for the values returned by the function to the calling program. Note that a function can take zero or more input arguments and return zero or more values. Here are some example function definitions:

```
function y=square(x);
function av=average(x1,x2,x3,x4,x5);
function printvalue(A);
function B=readvalue();
function [mean,sttdev]=analyse(tab);
```

Following the function line you should write a one line comment that summarises what the function does. For example:

% square(x) returns the square of the argument x

This line is printed out if the user types

lookfor *funcname*;

in the command window. All the comment lines between the function definition and the first executable statement are printed out when the user types

help *funcname*;

in the command window. Use this facility to provide some help information to the users of your function.

The body of your function will normally perform some computation based on the input arguments and end by assigning some values to the output arguments. When the function is called from another program, whatever values are supplied to the function are copied into the dummy input arguments, then the function is executed, then the values of the output dummy arguments are inserted into the calculation in the calling program. It is good practice to end each function with the returnstatement to remind you that execution returns to the calling program at this point.

```
function y=cube(x)
% cube(x) returns the cube of x
y = x * x * x;
return;
```

```
a=10;
b=cube(a);
disp(b);            % \
disp(cube(a));      %  All display 1000
disp(cube(10));     % /
```

It is good practice to pass all the information you need for a function through the list of input arguments and to receive all the processed results through the output arguments. Although this requires a lot of copying, MATLAB does this quite efficiently. Sometimes however, you may have a number of functions that all require access to the same table of data, and you don't want to keep copying the table into the function and then copying the changes back into your program. Imagine if the table had a million elements! Under these circumstances you can declare variables as 'global'. This means that they can be accessed both inside your program and inside a function without having to pass the variable as a function argument. Here is an example:

```
function initialisetable(num)
% initialise global variable TAB to all the same value
global TAB;
TAB=num*ones(size(TAB));
```

```
% main program
global TAB;
TAB=zeros(1,100);
initialisetable(5);
```

You can also write functions which take a variable number of arguments. In fact MATLAB allows any function to be called with fewer arguments than the definition, so it is a good idea to always check the number of arguments supplied. The built in variable 'nargin' contains the number of input arguments actually supplied, and 'nargout' contains the number of output arguments. You can use the built in function 'error()' to report an error if the number of arguments is incorrect. For example:

```
function m=average(x,y)
if (nargin!=2)
error('two arguments needed in average()');
end
```

We'll meet the if statement in the next lesson.

M-FILES

MATLAB allows writing two kinds of program files −

- **Scripts** − script files are program files with **.m extension**. In these files, you write series of commands, which you want to execute together. Scripts do not accept inputs and do not return any outputs. They operate on data in the workspace.

- **Functions** − functions files are also program files with **.m extension**. Functions can accept inputs and return outputs. Internal variables are local to the function.

You can use the MATLAB editor or any other text editor to create your **.m**files. In this section, we will discuss the script files. A script file contains multiple sequential lines of MATLAB commands and function calls. You can run a script by typing its name at the command line.

Creating and Running Script File

To create scripts files, you need to use a text editor. You can open the MATLAB editor in two ways:

- Using the command prompt
- Using the IDE

If you are using the command prompt, type **edit** in the command prompt. This will open the editor. You can directly type **edit** and then the filename (with .m extension)

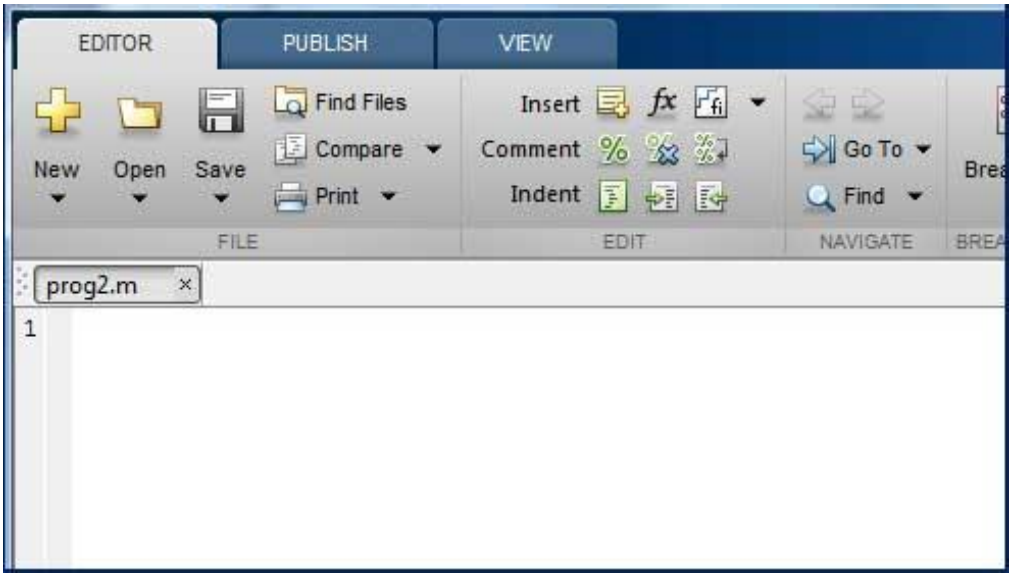
```
edit
Or
edit <filename>
```

The above command will create the file in default MATLAB directory. If you want to store all program files in a specific folder, then you will have to provide the entire path.

Let us create a folder named progs. Type the following commands at the command prompt (>>):

```
mkdir progs    % create directory progs under default directory
chdir progs    % changing the current directory to progs
edit  prog1.m  % creating an m file named prog1.m
```

If you are creating the file for first time, MATLAB prompts you to confirm it. Click Yes.



Alternatively, if you are using the IDE, choose NEW -> Script. This also opens the editor and creates a file named Untitled. You can name and save the file after typing the code.

Type the following code in the editor −

```
NoOfStudents = 6000;
TeachingStaff = 150;
NonTeachingStaff = 20;
Total = NoOfStudents + TeachingStaff ...
  + NonTeachingStaff;
disp(Total);
```

After creating and saving the file, you can run it in two ways −

- Clicking the **Run** button on the editor window or
- Just typing the filename (without extension) in the command prompt: >> prog1

The command window prompt displays the result −

```
6170
```

Example

Create a script file, and type the following code −

```
a = 5; b = 7;
c = a + b
d = c + sin(b)
e = 5 * d
f = exp(-d)
```

When the above code is compiled and executed, it produces the following result −

```
c =  12
d =  12.657
e =  63.285
f =   3.1852e-06
```

FORMATTED CONSOLE INPUT-OUTPUT

You can control the exact way in which values are printed to the screen with the 'fprintf()' function (fprintf= "file print formatted"). This function takes one argument repesenting the formatting instructions, followed by a list of values to be printed. Embedded within the format string are 'percent commands' which control where and how the values are to be written. Here are some examples:

```
fprintf('The answer is %g seconds.\n',nsec);
fprintf('Day of the week = %s\n',dayofweek([7 12 1941]));
fprintf('Mean=%.3f ± %.4f\n',mean,stddev);
```

The command %g represents a general real number, %f means a fixed point number, %d a decimal integer, and %s a string. You can put numeric values between the '%' and the letter to control the field width and the number of digits after the decimal point. For example (□=space):

fprintf('%5g',10)	□□□10
fprintf('%10.4f',123.456)	□□123.4560
fprintf('%10s', 'fred')	□□□□□□fred

You can input a value or a string from the command line with the 'input()' function. This has two forms depending on whether you want to input a number or a string:

```
yval=input('Enter a number: ');
name=input('Enter your name: ', 's');
```

Input and Output Commands

MATLAB provides the following input and output related commands −

Command	Purpose
disp	Displays contents of an array or string.
fscanf	Read formatted data from a file.
format	Controls screen-display format.

fprintf	Performs formatted writes to screen or file.
input	Displays prompts and waits for input.
;	Suppresses screen printing.

The **fscanf** and **fprintf** commands behave like C scanf and printf functions. They support the following format codes

STRING HANDLING

Simple strings are stored as tables with one row and a number of columns: one column per character. You can concatenate any table or strings simply by making the contents part of one table. For example:

```
str1='Hello';
str2='Mark ';
str=[str1 ' ' str2];
```

You can convert numbers to strings using the 'sprintf()' function, which operates analogously to the fprintf() function but outputs to a string rather than to the screen.

```
str=sprintf('%10.4f',123.45);
```

The 'abs()' function can be used to find the standard character codes for a string:

```
disp(abs('Mark'));
   77    97   114   107
```

The 'char()' function can be used to convert character codes back to a string:

```
disp(char([77 97 114 107]));
Mark
```

The 'eval()' function can be used to evaluate an expression stored in a string. This allows you to execute expressions typed in by the user:

```
expr=input('Enter an expression (e.g. "2+3*4") : ', 's');
disp(eval(expr));
```

Creating a character string is quite simple in MATLAB. In fact, we have used it many times. For example, you type the following in the command prompt −

```
my_string = 'Tutorials Point'
```

MATLAB will execute the above statement and return the following result −

```
my_string = Tutorials Point
```

MATLAB considers all variables as arrays, and strings are considered as character arrays. Let us use the **whos** command to check the variable created above −

```
whos
```

MATLAB will execute the above statement and return the following result −

```
Name          Size            Bytes  Class    Attributes
my_string     1x16               32  char
```

Interestingly, you can use numeric conversion functions like **uint8** or **uint16**to convert the characters in the string to their numeric codes. The **char**function converts the integer vector back to characters −

Example

Create a script file and type the following code into it −

```
my_string = 'Tutorial''s Point';
str_ascii = uint8(my_string)        % 8-bit ascii values
str_back_to_char= char(str_ascii)
str_16bit = uint16(my_string)       % 16-bit ascii values
str_back_to_char = char(str_16bit)
```

When you run the file, it displays the following result −

```
str_ascii =

  84  117  116  111  114  105   97  108   39  115   32   80  111  105  110  116
```

```
str_back_to_char = Tutorial's Point
str_16bit =

  84  117  116  111  114  105   97  108   39  115   32   80  111  105  110  116

str_back_to_char = Tutorial's Point
```

Rectangular Character Array

The strings we have discussed so far are one-dimensional character arrays; however, we need to store more than that. We need to store more dimensional textual data in our program. This is achieved by creating rectangular character arrays.

Simplest way of creating a rectangular character array is by concatenating two or more one-dimensional character arrays, either vertically or horizontally as required.

You can combine strings vertically in either of the following ways −

- Using the MATLAB concatenation operator **[]** and separating each row with a semicolon (;). Please note that in this method each row must contain the same number of characters. For strings with different lengths, you should pad with space characters as needed.

- Using the **char** function. If the strings are of different lengths, char pads the shorter strings with trailing blanks so that each row has the same number of characters.

Example

Create a script file and type the following code into it −

```
doc_profile = ['Zara Ali                    '; ...
               'Sr. Surgeon                 '; ...
               'R N Tagore Cardiology Research Center']
doc_profile = char('Zara Ali', 'Sr. Surgeon', ...
                   'RN Tagore Cardiology Research Center')
```

When you run the file, it displays the following result −

```
doc_profile =
Zara Ali
```

```
Sr. Surgeon
R N Tagore Cardiology Research Center
doc_profile =
Zara Ali
Sr. Surgeon
RN Tagore Cardiology Research Center
```

You can combine strings horizontally in either of the following ways −

- Using the MATLAB concatenation operator, **[]** and separating the input strings with a comma or a space. This method preserves any trailing spaces in the input arrays.
- Using the string concatenation function, **strcat**. This method removes trailing spaces in the inputs.

Example

Create a script file and type the following code into it −

```
name =     'Zara Ali                    ';
position = 'Sr. Surgeon                   ';
worksAt =  'R N Tagore Cardiology Research Center';
profile = [name ', ' position ', ' worksAt]
profile = strcat(name, ', ', position, ', ', worksAt)
```

When you run the file, it displays the following result −

```
profile = Zara Ali                    , Sr. Surgeon                    , R N Tagore Cardiology
Research Center
profile = Zara Ali,Sr. Surgeon,R N Tagore Cardiology Research Center
```

Combining Strings into a Cell Array

From our previous discussion, it is clear that combining strings with different lengths could be a pain as all strings in the array has to be of the same length. We have used blank spaces at the end of strings to equalize their length.

However, a more efficient way to combine the strings is to convert the resulting array into a cell array.

MATLAB cell array can hold different sizes and types of data in an array. Cell arrays provide a more flexible way to store strings of varying length.

The **cellstr** function converts a character array into a cell array of strings.

Example

Create a script file and type the following code into it −

```
name =     'Zara Ali                    ';
position = 'Sr. Surgeon                   ';
worksAt =  'R N Tagore Cardiology Research Center';
profile = char(name, position, worksAt);
profile = cellstr(profile);
disp(profile)
```

When you run the file, it displays the following result −

```
{
  [1,1] = Zara Ali
  [2,1] = Sr. Surgeon
  [3,1] = R N Tagore Cardiology Research Center
}
```

String Functions in MATLAB

MATLAB provides numerous string functions creating, combining, parsing, comparing and manipulating strings.

Following table provides brief description of the string functions in MATLAB −

Function	**Purpose**
Functions for storing text in character arrays, combine character arrays, etc.	
blanks	Create string of blank characters

cellstr	Create cell array of strings from character array
char	Convert to character array (string)
iscellstr	Determine whether input is cell array of strings
ischar	Determine whether item is character array
sprintf	Format data into string
strcat	Concatenate strings horizontally
strjoin	Join strings in cell array into single string
Functions for identifying parts of strings, find and replace substrings	
ischar	Determine whether item is character array
isletter	Array elements that are alphabetic letters
isspace	Array elements that are space characters
isstrprop	Determine whether string is of specified category
sscanf	Read formatted data from string
strfind	Find one string within another

strrep	Find and replace substring
strsplit	Split string at specified delimiter
strtok	Selected parts of string
validatestring	Check validity of text string
symvar	Determine symbolic variables in expression
regexp	Match regular expression (case sensitive)
regexpi	Match regular expression (case insensitive)
regexprep	Replace string using regular expression
regexptranslate	Translate string into regular expression
Functions for string comparison	
strcmp	Compare strings (case sensitive)
strcmpi	Compare strings (case insensitive)
strncmp	Compare first n characters of strings (case sensitive)
strncmpi	Compare first n characters of strings (case insensitive)

Functions for changing string to upper- or lowercase, creating or removing white space	
deblank	Strip trailing blanks from end of string
strtrim	Remove leading and trailing white space from string
lower	Convert string to lowercase
upper	Convert string to uppercase
strjust	Justify character array

Examples

The following examples illustrate some of the above-mentioned string functions −

FORMATTING STRINGS

Create a script file and type the following code into it −

```
A = pi*1000*ones(1,5);
sprintf(' %f \n %.2f \n %+.2f \n %12.2f \n %012.2f \n', A)
```

When you run the file, it displays the following result −

```
ans =  3141.592654
 3141.59
 +3141.59
     3141.59
 000003141.59
```

JOINING STRINGS

Create a script file and type the following code into it −

```
%cell array of strings
str_array = {'red','blue','green', 'yellow', 'orange'};
```

```
% Join strings in cell array into single string
str1 = strjoin(str_array, "-")
str2 = strjoin(str_array, ",")
```

When you run the file, it displays the following result −

```
str1 = red-blue-green-yellow-orange
str2 = red,blue,green,yellow,orange
```

FINDING AND REPLACING STRINGS

Create a script file and type the following code into it −

```
students = {'Zara Ali', 'Neha Bhatnagar', ...
        'Monica Malik', 'Madhu Gautam', ...
        'Madhu Sharma', 'Bhawna Sharma',...
        'Nuha Ali', 'Reva Dutta', ...
        'Sunaina Ali', 'Sofia Kabir'};

% The strrep function searches and replaces sub-string.
new_student = strrep(students(8), 'Reva', 'Poulomi')
% Display first names
first_names = strtok(students)
```

When you run the file, it displays the following result −

```
new_student =
{
  [1,1] = Poulomi Dutta
}
first_names =
{
  [1,1] = Zara
  [1,2] = Neha
  [1,3] = Monica
  [1,4] = Madhu
  [1,5] = Madhu
  [1,6] = Bhawna
```

```
  [1,7] = Nuha
  [1,8] = Reva
  [1,9] = Sunaina
  [1,10] = Sofia
}
```

COMPARING STRINGS

Create a script file and type the following code into it −

```
str1 = 'This is test'
str2 = 'This is text'
if (strcmp(str1, str2))
 sprintf('%s and %s are equal', str1, str2)
else
 sprintf('%s and %s are not equal', str1, str2)
end
```

When you run the file, it displays the following result −

```
str1 = This is test
str2 = This is text
ans = This is test and This is text are not equal
```

PART-B(2 MARKS)

POSSIBLE QUESTIONS

1. What is the type of program files that MATLAB allows to write?
2. What is an M-File?
3. What is String handling?
4. What is graph plots?
5. What is a function?
6. How to generate wave forms in MATLAB?

PART-C(6 MARKS)

POSSIBLE QUESTIONS

1. Explain about Basic Plotting in detail.
2. Explain in detail about M-Files.
3. Explain about Generating wave forms, Sound, replay, load and in detail.
4. Explain in detail about String handling.
5. Explain in detail about Procedures and Functions.
6. Explain in detail about Formatted Console Input- Output.

KARPAGAM ACADEMY OF HIGHER EDUCATION
Department of Computer Science
II B.Sc(CS) (BATCH 2016-2018)
Programming In MATLAB
PART-A OBJECTIVE TYPE/ MULTIPLE CHOICE QUESTI
ONLINE EXAMINATIONS **ONE MAR**

S.No	Question	Option 1
1	To add a comment to the mfile, the MATLAB command is	%
2	When used in the fprintf command, the %g is used as the	single character display
3	When used in the fprintf command, the \n is used to	add a space between any two characters
4	The dot (.) in MATLAB is used for	element to element mathematical operating
5	the standard inputs for the loglog command are	(log(x), y)
6	The MATLAB command to make a plot is	figure
7	The command to add text to the x axis of a plot is	xtitle
8	To add a superscript, use the charater(s)	\^
9	The command to add a legend to a plot is	plot,legend
10	The LineWidth command	adjusts the overall size of the figure font
11	The command \bf	creates bold font for all subsequent text
12	The _____ function rounds x to the nearest integer twords zero	ceil(x)
13	The _____ function rounds x to the nearest integer	ceil(x)
14	When the _____ function is executed, MATLAB opens the Figure window and displays the plot in that window	edit
15	The _____ function plots both x and y data on logarithmic axes	semilogx
16	The _____ function plots x data on linear axes and y data on logarithmix axes	semilogx
17	The basic building block in MATLAB is _____	matrix
18	The _____ command clears the screen	clc
19	The _____ command clears the figure window	clc
20	the _____ returns tangent of an angle given in degrees	tang
21	If a step size is not specified , + ___is taken as default value of the step size	2
22	The _____ gives the number of elements in a row/column vector	len(x)

23	The _____ command returns the number of elements of the matrix in each dimentsions	len(x)
24	The _____ function concatenates a list of arrays along a specified dimension	join
25	the _____ function gives the minimum value in row/column vector	min
26	The _____ function gives the maximum value in row/column vector	min
27	The _____ command is used to display only the subset of the data	axes
28	The _____ command sets the axis increments to be equal on both axes	axis normal
29	The _____ command makes the current axis box square	axis normal
30	the _____ command cancels the effect of axis equal and axis sqaure	axis normal
31	When a _____ command is used the additional plots will be laid on top of the previously existing plots	hold on
32	A _____ command switches plotting behaviour back to the default situation in which a new plot repalces the previous one	hold on
33	Each figure is identifed by the _____	window number
34	the current figure is selected with the _____ fucntion	window
35	The function _____ returns the number of the current figure	acf
36	A _____ is a special sequence of characters that ells the MATLAB interperter top change its behaviour	stream modifier
37	_____ is a stream modifier which replces the normal font	\rm
38	_____ plots data in polar corodinates	pole
39	Functions receive input data from the program tha tinvokes them through a list of variables called an _____ argument list	input
40	_____ are just collections of MATLAB statements that are stored in a file	function files
41	A MATLAB function is a special type of _____ that runs in its own independent workspace	G file
42	The _____ statement marks the beginning of the function	structure
43	A function is invoked by naming it in an expression together with a list of _____ arguments	formal
44	The _____ statement is used to terminate the function	stop
45	The first comment line after the function statement is called the _____ comment line	H1
46	MATLAB programs communicate with their functionsusing a _____ scheme	pass by value
47	_____ function returns the number of actual input arguments that were used to call the function	nargin
48	_____ fucntion returns the number of actual output arguments that were used to call the function	nargin
49	_____ funtion returns a standard error message if a function is called with too few or too many arguments	nargin
50	_____ displays warning messasge and continue function excution	nargin
51	_____ is a special type of memory tha tcan be accessed from any workspace	static memory
52	_____ provides a way to share data between functions	static memory
53	A _____ variable is declared with the global statement	local

54	_____ is a special type of memory that can be accessed only within the function, but is preserved unchanged between calls to the function	static memory
55	_____ are functions whose input arguments include the names of other functions	function files
56	_____ function locates a zero of the function tha tis apssed to it.	fempty
57	Variable can be converte dfrom double data type to char data type using _____ function	char
58	The easiest way to produce two dimensional character arrays is with the _____ fucntion	int
59	_____ function can be used to remove extra blanks from a string when it is extracted from an array	remove
60	Two dimensional character arrays can also be created with function _____	string
61	_____ functions concatenates two or more strings ignoring trailing blanks	strrev
62	_____ function determines if two strings are identical	strrev
63	_____ is a type of polar plot in which each value represented by an arrow whose length is proportional to its value	bar plot
64	_____ function determines if the first n characters of two strings are identical	strncmp
65	_____ determines if the first n characters of two strings are identical ignoring cases	strncmp
66	_____ function determines if a character is a letter	isalpha
67	A _____ plot is a plot in which each dat avalue is represented by a marker and a line connecting the marker vertically to the x aixs	stair
68	_____ finds matches for string	strfind
69	_____ function replaces onestring with another	strfind
70	_____ function is used to justify the string	strjust
71	A ____ plot is a plot in which each point is represented by a vertical bar or horizontal bar	stair
72	_____ functions removes any extra leading and trailing whitespace from a string	deblank
73	_____ function converts a double value into a string	num2str
74	dec2hex converts a _____ value into corresponding hexadecimal string	integer
75	MATLAB function _____ converts an array to a string that MATLAB can evaluate	mat2int
76	In _____ function the output goes into a character string instead of the command window	fprintf

ONS
RK QUESTIONS

Option 2	Option 3	Option 4		
;	comment(' ')	&		
fixed point display	string notation display	default number display		
add a line space (enter key)	place a number into the comment	clear the comment		
ending a command	naming a figure	requesting a colorful candy		
(x,y)	(log(x),log(y))	(log10(x),log10(y))		
fit	plot	pplot		
label,x	xlabel	xtext		
^	\super	\s		
legend,plot	legend	leg		
adjusts the size of the plotted points	changes the size of the figure border	changes the thickness of plotted lines		
stands for best friend	creates bold font for all preceding text	creates a new line in the title of the plot		
fix(x)	floor(x)	round(x)		
fix(x)	floor(x)	round(x)		
figure	plotting	plot		
semilogy	loglog	log		
semilogy	loglog	log		
vector	scalar	functions		
clr	cls	cle		
clf	cls	cle		
tand	tan	tan2		
1	3	4		
size(x)	length(x)	none		

length(x)	size(x)	none		
cat	rand	joined		
least	max	minum		
least	max	minum		
axis	plot	plotting		
axis square	axis on	axis equal		
axis square	axis on	axis equal		
axis square	axis on	axis equal		
hold off	holded on	none		
hold off	holded on	none		
screen number	figure number	picture number		
figure	subplot	plotting		
gaf	gcf	agf		
modifier	online modifier	file modifier		
\rrf	\rf	\fr		
polar	plot	poles		
output	result	fucntion		
script files	legal files	none		
M file	MM file	MX file		
function	parameters	none		
informal	argument	actual		
finish	end	none		
L1	G1	E1		
pass by no values	pass by parameters	none		
nargout	nargchk	erro		
nargout	nargchk	erro		
nargout	nargchk	erro		
nargout	nargchk	warning		
dynamic memory	global memory	random memory		
dynamic memory	global memory	random memory		
global	persistent	protected		

dynamic memory	global memory	persistent memory		
function functions	sub function	recursive function		
fzero	fnull	fone		
int	double	string		
char	double	string		
deblank	trim	delete		
character	strvcat	strrev		
strvcat	strcat	strcon		
strcmp	strncmp	stricmp		
compass plot	pie plot	stem plot		
strcmp	strcmpi	stricmp		
strcmp	strncmpi	stricmp		
isletter	ischar	isstring		
stem	bar	pie		
strmatch	strrep	strrrev		
strmatch	strrep	strrrev		
strmatch	strrep	strrrev		
stem	bar	pie		
strtrim	strrev	strrep		
int2str	str2num	none of the above		
double	long int	none of the above		
str2mat	mat2str	none of the above		
sprintf	printf	print		

Answers
%
default number display
add a line space (enter key)
element to element mathematical operating
(x,y)
plot
xlabel
^
legend
changes the thickness of plotted lines
creates bold font for all subsequent text
fix(x)
round(x)
plot
loglog
semilogy
matrix
clc
clf
tand
1
length(x)

size(x)
cat
min
max
axis
axis equal
axis square
axis normal
hold on
hold off
figure number
figure
gcf
stream modifier
\rm
polar
input
script files
M file
function
actual
end
H1
pass by value
nargin
nargout
nargchk
warning
global memory
global memory
global

persistent memory
function functions
fzero
char
char
deblank
strvcat
strcat
strcmp
compass plot
strncmp
strncmpi
isletter
stem
strmatch
strrep
strjust
bar
strtrim
num2str
double
mat2str
sprintf

UNIT IV

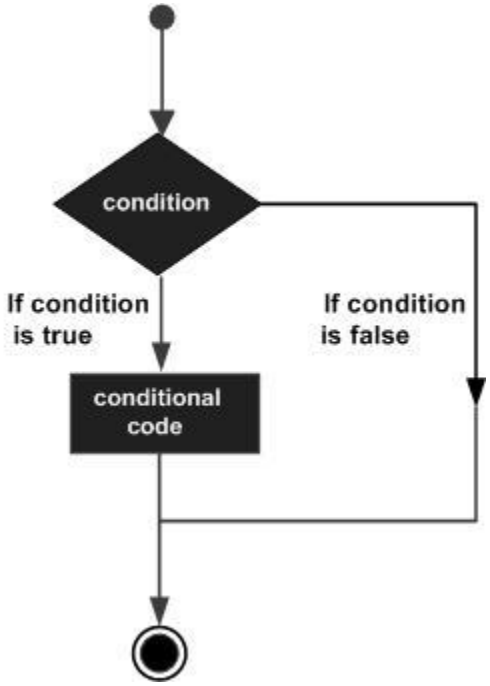
SYLLABUS

Control Statements: Conditional statements: If, Else, Else-if, Repetition statements: While, for loop

CONDITIONAL STATEMENTS

Decision making structures require that the programmer should specify one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.

Following is the general form of a typical decision making structure found in most of the programming languages −



CONDITIONAL STATEMENTS

MATLAB provides following types of decision making statements. Click the following links to check their detail −

Statement	Description
if ... end statement	An **if ... end statement** consists of a boolean expression followed by one or more statements.
if...else...end statement	An **if statement** can be followed by an optional **else statement**, which executes when the boolean expression is false.
If... elseif...elseif...else...end statements	An **if** statement can be followed by one (or more) optional **elseif...** and an **else** statement, which is very useful to test various conditions.
nested if statements	You can use one **if** or **elseif** statement inside another **if** or **elseif** statement(s).
switch statement	A **switch** statement allows a variable to be tested for equality against a list of values.
nested switch statements	You can use one **switch** statement inside another **switch** statement(s).

If end

An **if ... end** statement consists of an **if** statement and a boolean expression followed by one or more statements. It is delimited by the **end** statement.

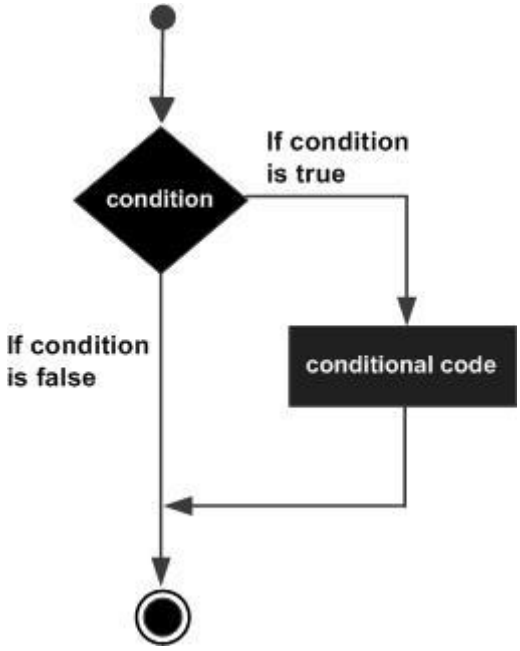
Syntax

The syntax of an if statement in MATLAB is −

```
if <expression>
% statement(s) will execute if the boolean expression is true
<statements>
end
```

If the expression evaluates to true, then the block of code inside the if statement will be executed. If the expression evaluates to false, then the first set of code after the end statement will be executed.

Flow Diagram



Example

Create a script file and type the following code −

```
a = 10;
% check the condition using if statement
  if a < 20
  % if condition is true then print the following
    fprintf('a is less than 20\n' );
```

```
  end
fprintf('value of a is : %d\n', a);
```

When you run the file, it displays the following result −

```
a is less than 20
value of a is : 10
```

If else end

An if statement can be followed by an optional else statement, which executes when the expression is false.

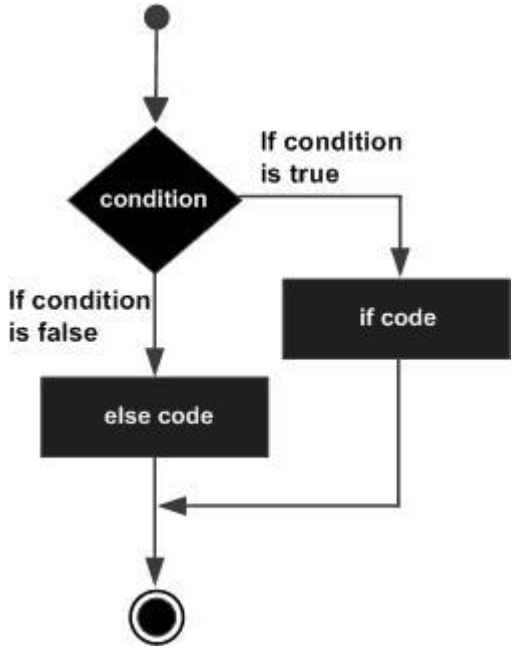
Syntax

The syntax of an if...else statement in MATLAB is −

```
if <expression>
% statement(s) will execute if the boolean expression is true
<statement(s)>
else
<statement(s)>
% statement(s) will execute if the boolean expression is false
end
```

If the boolean expression evaluates to true, then the if block of code will be executed, otherwise else block of code will be executed.

Flow Diagram



Example

Create a script file and type the following code −

```
a = 100;
% check the boolean condition
  if a < 20
    % if condition is true then print the following
    fprintf('a is less than 20\n' );
  else
    % if condition is false then print the following
    fprintf('a is not less than 20\n' );
  end
  fprintf('value of a is : %d\n', a);
```

When the above code is compiled and executed, it produces the following result −

```
a is not less than 20
```

```
value of a is : 100
```

If elseif elseif else end statements

An **if** statement can be followed by one (or more) optional **elseif...** and an **else** statement, which is very useful to test various conditions.

When using if... elseif...else statements, there are few points to keep in mind:

- An if can have zero or one else's and it must come after any elseif's.
- An if can have zero to many elseif's and they must come before the else.
- Once an else if succeeds, none of the remaining elseif's or else's will be tested.

Syntax

```
if <expression 1>
% Executes when the expression 1 is true
<statement(s)>

elseif <expression 2>
% Executes when the boolean expression 2 is true
<statement(s)>

Elseif <expression 3>
% Executes when the boolean expression 3 is true
<statement(s)>

else
%  executes when the none of the above condition is true
<statement(s)>
end
```

Example

Create a script file and type the following code in it −

```
a = 100;
%check the boolean condition
  if a == 10
    % if condition is true then print the following
    fprintf('Value of a is 10\n' );
  elseif( a == 20 )
    % if else if condition is true
    fprintf('Value of a is 20\n' );
  elseif a == 30
    % if else if condition is true
    fprintf('Value of a is 30\n' );
  else
    % if none of the conditions is true '
    fprintf('None of the values are matching\n');
  fprintf('Exact value of a is: %d\n', a );
  end
```

When the above code is compiled and executed, it produces the following result −

```
None of the values are matching
Exact value of a is: 100
```

Nested If Statements

It is always legal in MATLAB to nest if-else statements which means you can use one if or elseif statement inside another if or elseif statement(s).

Syntax

The syntax for a nested if statement is as follows −

```
if <expression 1>
% Executes when the boolean expression 1 is true
   if <expression 2>
      % Executes when the boolean expression 2 is true
  end
end
```

You can nest elseif...else in the similar way as you have nested if statement.

Example

Create a script file and type the following code in it −

```
a = 100;
b = 200;
   % check the boolean condition
   if( a == 100 )

      % if condition is true then check the following
      if( b == 200 )

         % if condition is true then print the following
         fprintf('Value of a is 100 and b is 200\n' );
      end

   end
   fprintf('Exact value of a is : %d\n', a );
   fprintf('Exact value of b is : %d\n', b );
```

When you run the file, it displays −

```
Value of a is 100 and b is 200
Exact value of a is : 100
```

```
Exact value of b is : 200
```

Switch Statements

A switch block conditionally executes one set of statements from several choices. Each choice is covered by a case statement.

An evaluated switch_expression is a scalar or string.

An evaluated case_expression is a scalar, a string or a cell array of scalars or strings.

The switch block tests each case until one of the cases is true. A case is true when −

- For numbers, **eq(case_expression,switch_expression)**.
- For strings, **strcmp(case_expression,switch_expression)**.
- For objects that support the **eq(case_expression,switch_expression)**.
- For a cell array case_expression, at least one of the elements of the cell array matches switch_expression, as defined above for numbers, strings and objects.

When a case is true, MATLAB executes the corresponding statements and then exits the switch block.

The **otherwise** block is optional and executes only when no case is true.

Syntax

The syntax of switch statement in MATLAB is −

```
switch <switch_expression>
  case <case_expression>
    <statements>
  case <case_expression>
    <statements>
    ...
    ...
```

```
  otherwise
    <statements>
end
```

Example

Create a script file and type the following code in it −

```
grade = 'B';
  switch(grade)
  case 'A'
    fprintf('Excellent!\n' );
  case 'B'
    fprintf('Well done\n' );
  case 'C'
    fprintf('Well done\n' );
  case 'D'
    fprintf('You passed\n' );

  case 'F'
    fprintf('Better try again\n' );

  otherwise
    fprintf('Invalid grade\n' );
  end
```

When you run the file, it displays −

```
Well done
```

Nested Switch statements

It is possible to have a switch as part of the statement sequence of an outer switch. Even if the case constants of the inner and outer switch contain common values, no conflicts will arise.

Syntax

The syntax for a nested switch statement is as follows −

```
switch(ch1)
  case 'A'
  fprintf('This A is part of outer switch');
    switch(ch2)
      case 'A'
      fprintf('This A is part of inner switch' );

      case 'B'
      fprintf('This B is part of inner switch' );
    end
  case 'B'
  fprintf('This B is part of outer switch' );
end
```

Example

Create a script file and type the following code in it −

```
a = 100;
b = 200;
switch(a)
  case 100
  fprintf('This is part of outer switch %d\n', a );

  switch(b)
    case 200
```

```
    fprintf('This is part of inner switch %d\n', a );
  end
end
fprintf('Exact value of a is : %d\n', a );
fprintf('Exact value of b is : %d\n', b );
```

When you run the file, it displays −

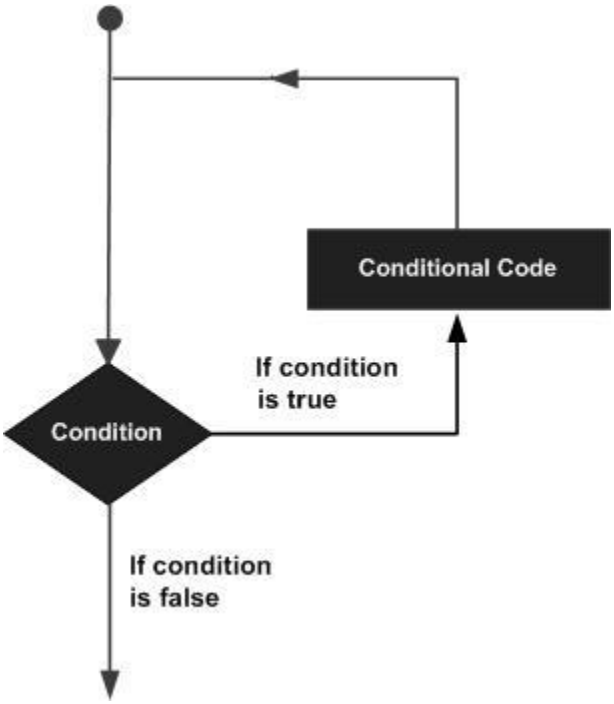
```
This is part of outer switch 100
This is part of inner switch 100
Exact value of a is : 100
Exact value of b is : 200
```

REPETITION STATEMENTS

There may be a situation when you need to execute a block of code several number of times. In general, statements are executed sequentially. The first statement in a function is executed first, followed by the second, and so on.

Programming languages provide various control structures that allow for more complicated execution paths.

A loop statement allows us to execute a statement or group of statements multiple times and following is the general form of a loop statement in most of the programming languages −



MATLAB provides following types of loops to handle looping requirements. Click the following links to check their detail −

Loop Type	Description
while loop	Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.
for loop	Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.
nested loops	You can use one or more loops inside any another loop.

Loop Control Statements

Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.

MATLAB supports the following control statements. Click the following links to check their detail.

Control Statement	Description
break statement	Terminates the **loop** statement and transfers execution to the statement immediately following the loop.
continue statement	Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.

While Loop

The while loop repeatedly executes statements while condition is true.

Syntax

The syntax of a while loop in MATLAB is −

```
while <expression>
  <statements>
end
```

The while loop repeatedly executes program statement(s) as long as the expression remains true.

An expression is true when the result is nonempty and contains all nonzero elements (logical or real numeric). Otherwise, the expression is false.

Example

Create a script file and type the following code −

```
a = 10;
% while loop execution
while( a < 20 )
  fprintf('value of a: %d\n', a);
```

```
  a = a + 1;
end
```

When you run the file, it displays the following result −

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

For Loop

A **for loop** is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.

Syntax

The syntax of a **for loop** in MATLAB is −

```
for index = values
   <program statements>
         ...
end
```

values has one of the following forms −

Format	Description
initval:endval	increments the index variable from *initval* to *endval* by 1, and repeats execution of *program statements* until *index* is greater than *endval*.

initval:step:endval	increments *index* by the value step on each iteration, or decrements when step is negative.
valArray	creates a column vector *index* from subsequent columns of array *valArray* on each iteration. For example, on the first iteration, index = valArray(:,1). The loop executes for a maximum of n times, where n is the number of columns of *valArray*, given by numel(valArray, 1, :). The input *valArray*can be of any MATLAB data type, including a string, cell array, or struct.

Example 1

Create a script file and type the following code −

```
for a = 10:20
   fprintf('value of a: %d\n', a);
end
```

When you run the file, it displays the following result −

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
value of a: 20
```

Example 2

Create a script file and type the following code −

```
for a = [24,18,17,23,28]
   disp(a)
```

```
end
```

When you run the file, it displays the following result −

```
24

18

17

23

28
```

Nested Loop

MATLAB allows to use one loop inside another loop. Following section shows few examples to illustrate the concept.

Syntax

The syntax for a nested for loop statement in MATLAB is as follows −

```
for m = 1:j
  for n = 1:k
    <statements>;
  end
end
```

The syntax for a nested while loop statement in MATLAB is as follows −

```
while <expression1>
  while <expression2>
    <statements>
  end
end
```

Example

Let us use a nested for loop to display all the prime numbers from 1 to 100. Create a script file and type the following code −

```
for i=2:100
  for j=2:100
    if(~mod(i,j))
      break; % if factor found, not prime
    end
  end
  if(j > (i/j))
    fprintf('%d is prime\n', i);
  end
end
```

When you run the file, it displays the following result −

```
2 is prime
3 is prime
5 is prime
7 is prime
11 is prime
13 is prime
17 is prime
19 is prime
23 is prime
29 is prime
31 is prime
37 is prime
41 is prime
43 is prime
47 is prime
53 is prime
59 is prime
61 is prime
67 is prime
71 is prime
```

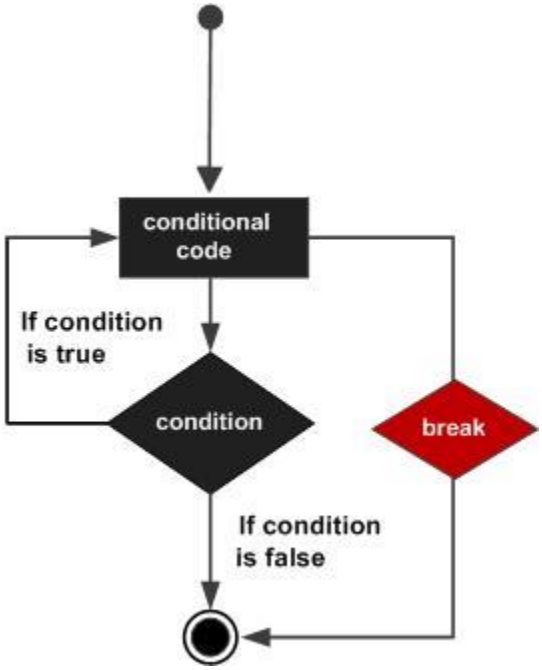
```
73 is prime
79 is prime
83 is prime
89 is prime
97 is prime
```

Break Statement

The break statement terminates execution of **for** or **while** loop. Statements in the loop that appear after the break statement are not executed.

In nested loops, break exits only from the loop in which it occurs. Control passes to the statement following the end of that loop.

Flow Diagram



Example

Create a script file and type the following code:

```
a = 10;
% while loop execution
```

```
while (a < 20 )
   fprintf('value of a: %d\n', a);
   a = a+1;
      if( a > 15)
         % terminate the loop using break statement
         break;
      end
end
```

When you run the file, it displays the following result:

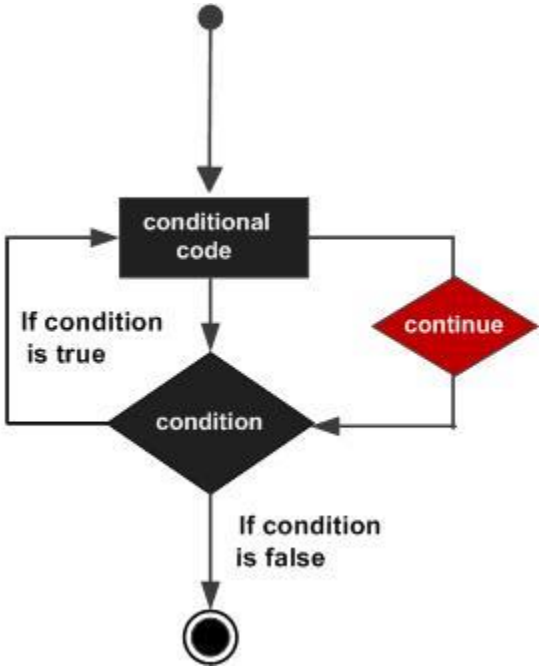
```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
```

Continue Statements

The continue statement is used for passing control to next iteration of for or while loop.

The continue statement in MATLAB works somewhat like the break statement. Instead of forcing termination, however, 'continue' forces the next iteration of the loop to take place, skipping any code in between.

Flow Diagram



Example

Create a script file and type the following code −

```
a = 10;
%while loop execution
while a < 20
  if a == 15
    % skip the iteration
    a = a + 1;
    continue;
  end
fprintf('value of a: %d\n', a);
a = a + 1;
end
```

When you run the file, it displays the following result −

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

PART-B(2 MARKS)

POSSIBLE QUESTIONS

1. What is If Statement and its syntax?
2. What are the types of loops does Matlab provides?
3. Write the syntax of while and for loop in MATLAB.

PART-C(6 MARKS)

POSSIBLE QUESTIONS

1. Explain Conditional statements with example.
2. Briefly describe about Repetition Statement.
3. Explain in detail while, for loop with example.

KARPAGAM ACADEMY OF HIGHER EDUCATION
Department of Computer Science
II B.Sc(CS) (BATCH 2016-2018)
Programming In MATLAB
PART-A OBJECTIVE TYPE/ MULTIPLE CHOICE QUESTIONS
ONLINE EXAMINATIONS **ONE MARK Q**

S.No	Question	Option 1
1	_____ permits a programmer to seelct a perticular code block to execute based on the value of a single integer, character or logical expression	if
2	The _____ construct is a special form branching construct designed to trap errors	try/catch
3	When an error occurs in the try block, it immediately excutes the stements in the _____ block	else
4	The statements in the _____ block will always be excuted	catch
5	The statements in the _____ block will only be executed of an error occurred in try block	catch
6	_____ are MATLAB constructs that permit us to execute a sequence of statements more than once	branches
7	A _____ loop is a block of statements that are repeated indefintiely as long as some condition is satisified	do while
8	The _____ loop is a loop that executes a block of statements a specified number of times	do while
9	The _____ of a for loop should not be modified anywhere within the body of the loop	body
10	In MATLAB, the process of replacing loops by vectorized statements is known as _____	scalarization
11	The JIT compiler helps to speed up the execution of _____ loops	do while
12	The _____ statement terminates the execution of a loop and passes control to the next statement after the end of the loop	break
13	the _____ statement terminates the current pass through the loop and return control to the top of the loop	break
14	If one loop is completely inside another one, the two loops are called _____ loops	double
15	when MATLAB encounters an _____ statement, it associates that statement with the innermost currently open construct	break
16	If _____ loops are nested, they should have independent loop index variables.	do while
17	If a break or continue statement appears inside a set of nested loops, then that statement refers to the _____ of the loops containing it.	innermost
18	Scalars and arrays of _____ data are created as the output of relational and logic oeprators	vectorization

19	_____ arrays can serve as a mask for arithmetic oeprations	logical
20	A _____ is an array that selects the elements of another array for use in an operation	set
21	when the increment value of the index is not mentioned, it is taken as _____ by default	2
22	In _____ structure, case can have multiple values	switch case
23	If the value of switch variable is _____, then it must be entered within single quotes	integer
24	_____ is used to terminate the program due to incorrect input and gives the error message	break
25	The ~= operator stands for	not equal to
26	The equivalent MATLAB expression for, A greater then or equal to B is	A>B
27	To end the body of a for-end loop the command is	end
28	How many logic tests can be used in a while-end loop?	A maximun of 1
29	The while-end loop will complete repetitions	until a logic statement is false
30	The while loop is a(an)	definite loop
31	The inline function is used to	accept inputs from the user
32	The for-end loop will repeat a segment of program	based on a vector counter.
33	The while-end loop is classified as a/an	definite loop
34	_____forces the next iteration of the loop to take place, skipping any code in between.	break
35	The _____ statement is used for passing control to next iteration of for or while loop	'continue'
36	The _____ statement terminates execution of for or while loop	for loop
37	Statements in the loop that appear after the _____ statement are not executed.	break
38	_____ is the use of one loop inside another loop	'continue'
39	_____ increments the index variable from initval to endval by 1	*initval:endval*
40	A _____ is a repetition control structure to execute a specific number of times.	Switch
41	A _____ block conditionally executes one set of statements from several choices	If

42	_____ statement can be followed by an optional else statement	Switch
43	For loop Begins with a ____ and ends with an end	If
44	_____ is a repetition statement	for
45	_____ is a decision control statement	for
46	The _____ loop repeatedly executes statements while condition is true.	for loop
47	In for loop _____ increments *index* by the value step on each iteration, or decrements when step is negative	*initval:endval*
48	In _____, break exits only from the loop in which it occurs.	nested loops

UESTIONS

Option 2	Option 3	Option 4			Answers
try	switch	if else			switch
switch	if	if else			try/catch
while	catch	none			catch
else	try	if else			try
else	try	if else			catch
loops	structures	union			loops
while	do	for			while
while	do	for			while
loop index	loop expression	none			loop index
vectorization	looping	branching			vectorization
while	for	if			for
continue	skip	end			break
continue	skip	end			continue
grouping	nesting	none			nesting
continue	end	skip			end
while	if	for			for
outermost	top	bottom			innermost
arithmetic	logical	none			logical

arithmetic	relational	none			logical
vector	mask	unmask			mask
1	3	4			1
if else	while	for			switch case
double	float	character			character
continue	error	none			error
equal to	less then or equal to	approximately equal to			not equal to
A>=B	A=>B	A>B,A=B			A>=B
over	fend	complete			end
A maximun of 2	A maximun of 3	as many as needed			as many as needed
until a logic statement is true	until a counter has expired	indefinitely			until a logic statement is false
indefinite loop	infinite loop	logic test			indefinite loop
concatenate functions	define a function	draw a line to seperate outputs			define a function
while a conditional statement is true	every time the enter key is pressed.	indefinitely.			based on a vector counter.
indefinite loop.	infinite loop.	ridiculous loop.			indefinite loop.
'continue'	nested loop	for loop			'continue'
nested loop	break	for loop			continue
break	'continue'	nested loop			break
for loop	nested loop	'continue'			break
nested loop	break	for loop			nested loop
initval:step:endval	*valArray*	for loop			*initval:endval*
If	for loop	For			for loop
Switch	For	for loop			Switch

for loop	If	For			If
for	Switch	for loop			For
else if	switch	nested if			For
while	nested loop	if			if
for	while	nested loop			While
initval:step:endval	*valArray*	For			*initval:step:endval*
for loop	For	Switch			nested loops

UNIT V

SYLLABUS

Manipulating Text: Writing to a text file, Reading from a text file, Randomising and sorting a list, searching a list. **GUI Interface**: Attaching buttons to actions, Getting Input, Getting Output

MANIPULATING TEXT

1. Writing to a text file

To save the results of some computation to a file in text format reqires the following steps:

a. Open a new file, or overwrite an old file, keeping a 'handle' for the file.

b. Print the values of expressions to the file, using the file handle

c. Close the file, using the file handle

The file handle is a just a variable which identifies the open file in your program. This allows you to have any number of files open at any one time.

```
% open file
fid = fopen('myfile.txt','wt');     % 'wt' means "write text"
if (fid < 0)
   error('could not open file "myfile.txt"');
end;
% write some stuff to file
for i=1:100
   fprintf(fid,'Number = %3d Square = %6d\n',i,i*i);
end;
% close the file
fclose(fid);
```

2. Reading from a text file

To read some results from a text file is straightforward if you just want to load the whole file into memory. This requires the following steps:

a. Open an existing file, keeping a 'handle' for the file.

b. Read expressions from the file into a single array, using the file handle

c. Close the file, using the file handle

The fscanf() function is the inverse of fprintf(). However it returns the values it reads as values in a matrix. You can control the 'shape' of the output matrix with a third argument.

```
A = fscanf(fid,"%g %g %g\n",[3,inf])      % A has 3 rows and 1 col per line
disp(A(1,1))        % display first value on first line
disp(A(1,2))        % display first value on second line
disp(A(2,1))        % display second value on first line
```

Thus to read back the data we saved above:

```
% open file
fid = fopen('myfile.txt','rt');     % 'rt' means "read text"
if (fid < 0)
   error('could not open file "myfile.txt"');
end;
% read from file into table with 2 rows and 1 column per line
tab = fscanf(fid,'Number = %d Square = %d\n',[2,inf]);
% close the file
fclose(fid);
rtab = tab';        % convert to 2 columns and 1 row per line
```

Reading a table of strings is more complex, since the strings have to be the same length. We can use the fgetl() function to get a line of text as characters, but we'll first need to find out the length of the longest string, then ensure all strings are the same length. Here is a complete function for loading a text file as a table of fixed-length strings:

```
function tab=readtextfile(filename)
% Read a text file into a matrix with one row per input line
% and with a fixed number of columns, set by the longest line.
% Each string is padded with NUL (ASCII 0) characters
%
% open the file for reading
ip = fopen(filename,'rt');          % 'rt' means read text
if (ip < 0)
    error('could not open file');   % just abort if error
end;
% find length of longest line
max=0;                              % record length of longest string
cnt=0;                              % record number of strings
s = fgetl(ip);                      % get a line
while (ischar(s))                   % while not end of file
   cnt = cnt+1;
   if (length(s) > max)             % keep record of longest
       max = length(s);
   end;
    s = fgetl(ip);                  % get next line
end;
% rewind the file to the beginning
frewind(ip);
% create an empty matrix of appropriate size
tab=char(zeros(cnt,max));           % fill with ASCII zeros
% load the strings for real
cnt=0;
s = fgetl(ip);
while (ischar(s))
   cnt = cnt+1;
   tab(cnt,1:length(s)) = s;        % slot into table
    s = fgetl(ip);
end;
% close the file and return
fclose(ip);
return;
```

Here is an example of its use:

```
% write some variable length strings to a file
op = fopen('weekdays.txt','wt');
fprintf(op,'Sunday\nMonday\nTuesday\nWednesday\n');
fprintf(op,'Thursday\nFriday\nSaturday\n');
fclose(op);
% read it into memory
tab = readtextfile('weekdays.txt');
% display it
disp(tab);
```

3. Randomising and sorting a list

Assuming we have a table of values, how can we randomise the order of the entries? A good way of achieving this is analogous to shuffling a pack of cards. We pick two positions in the pack, then swap over the cards at those two positions. We then just repeat this process enough times that each card is likely to be swapped at least once.

```
function rtab=randomise(tab)
% randomise the order of the rows in tab.
% columns are unaffected
[nrows,ncols]=size(tab);          % get size of input matrix
cnt = 10*nrows;                   % enough times
while (cnt > 0)
   pos1 = 1+fix(nrows*rand);      % get first random row
   pos2 = 1+fix(nrows*rand);      % get second random row
   tmp = tab(pos1,:);             % save first row
   tab(pos1,:) = tab(pos2,:);     % swap second into first
   tab(pos2,:) = tmp;             % move first into second
   cnt=cnt-1;
end;
rtab=tab;                         % return randomised table
return;
```

This function should take two rows and return –1 if the first row sorts earlier than the second, 1 if the second row sorts earlier than the first and 0 if there is no preference. Here is a case-independent comparison function:

```
function flag=comparenocase(str1,str2)

% compares two strings without regard to case
% returns –1, 0, 1 if str1 is less than, equal, greater than str2.
len1=length(str1);
len2=length(str2);
for i=1:min(len1,len2)
   c1 = str1(i);
   c2 = str2(i);
   if (('a' <= c1)&(c1 <= 'z'))
      c1 = char(abs(c1)-32);          % convert lower case to upper
   end;
   if (('a' <= c2)&(c2 <= 'z'))
      c2 = char(abs(c2)-32);          % convert lower case to upper
   end;
   if (c1 < c2)
      flag = -1;                  % str1 sorts earlier
      return;
   elseif (c2 < c1)
      flag = 1;                   % str2 sorts earlier
      return;
   end;
end;
% strings match up to length of shorter, so
if (len1 < len2)
   flag = -1;                     % str1 sorts earlier
elseif (len2 < len1)
   flag = 1;                      % str2 sorts earlier
else
   flag = 0;                      % no preference
end;
return;
```

Here is a sort function that might be used with this comparison function.

```
function stab=functionsortrows(tab,funcname)
% sorts the rows of the input table using the supplied
% function name to provide an ordering on pairs of rows
[nrows,ncols]=size(tab);
for i=2:nrows                     % sort each row into place
   j = i;
```

```
    tmp = tab(j,:);                 % save row
    % compare this row with higher rows to see where it goes
    while ((j > 1)&(feval(funcname,tmp,tab(j-1,:))<0))
       tab(j,:) = tab(j-1,:);       % shift higher rows down
       j = j - 1;
    end;
    tab(j,:) = tmp;                 % put in ordered place
end;
stab = tab;                         % return sorted table
return;
```

4. Searching a list

If the list is unordered, all we can do is run down the list testing each entry in turn. This function finds the index of a row in a table that contains (anywhere) the characters in the supplied match string:

```
function idx=findstring(tab,str)
% find the row index containing a matching string
% returns 0 if the string is not found
[nrows,ncols]=size(tab);
for idx=1:nrows
   matches = findstr(tab(idx,:),str);
   if (length(matches)>0)
      return;
   end;
end;
idx=0;
return;
```

The process can be much faster if the listed is sorted and we are searching for an exact match only. A so-called binary search is the fastest possible way of finding an item in a sorted list:

```
function idx=binarysearch(tab,val)
% returns the row index of val in sorted table tab
% returns 0 if val is not found
[nrows,ncols]=size(tab);
```

```
lo=1;
hi=nrows;
while (lo <= hi)
   idx = fix(lo+hi)/2;
   if (val < tab(idx,:))
      hi = idx - 1;
   elseif (val > tab(idx,:))
      lo = idx + 1;
   else
      return;
   end;
end;
idx=0;
return;
```

GUI INTERFACE

1. Elements of a Graphical User Interface

By a graphical user interface, we mean that we can give a MATLAB program the look and feel of a typical Windows application. The MATLAB GUI design system allows you to create applications consisting of one or more 'dialogs' containing typical 'controls' such as buttons, edit boxes, lists and pictures.

One of the important aspects of a Windows application that is unlike the kind of programs we have considered up to now is that they interact asynchronously with the user. The user can select any function of the program at any time. This means that you need to store the 'state' of your program in a set of variables and be prepared to execute any function based on the current state at any time.

The MATLAB GUI design system helps you in this by associating functions with each element of the dialog. Thus when you press a button, click on a menu, or enter a number in an edit box, you can arrange for a function in your program to be called. Your task is to program the actions related to that function, e.g. opening a file, playing a sound, or displaying the results of a calculation.

The most common controls are:

- Menu options. Selection calls up an operation by name.
- Push buttons. Clicking calls up some operation.

- Edit boxes. User can enter some text or numerical value.
- List boxes. User can choose among list of items.
- Figures. Program can display graphical results.
- Text. Program can display textual result.

You can use the controls themselves to store data or you can create a set of global variables.

2. How to build a simple dialogue

To start the design program type 'guide' at the MATLAB prompt. You are presented with a blank form upon which you can position controls. Choose a control from the palette and click and size the control on the page to position it. Each control is automatically given a name based on its type.

When the layout is complete, you can save the design to a '.fig' file. This will automatically create a matching '.m' program file which you can use to launch the application and store the code that is operated by the controls. It is not necessary to store all your code in the matching '.m' file; indeed it is a good idea to break up any large sections of code into its own function blocks stored in separate files. You will see that the layout designer builds a 'callback' function prototype in the program file for each control that provides input to the application. This function will be called automatically when that control is activated.

We can edit the properties of the controls on the layout editor by right-clicking on them and choosing 'Property Inspector'. In particular the 'String' property is used to store the default text for buttons, list boxes and edit boxes. The 'Tag' property is the name of the control; and until you are familiar with MATLAB, it is advisable not to change the default name. You can also use the Property Inspector to change the name of the dialog itself.

We can add menu options to your dialog with the 'Menu Editor'. If you leave the callback function entry as "%automatic", then the menu editor adds callback functions to your program for each menu item. Otherwise create your own callback function using existing ones as a model, and associate a call to the function with the menu item manually.

It is important to realise that the '.m' file associated with your application is executed afresh each time there is some event in the dialog. That is you must store the 'current state' of the program in global variables in the workspace, and not in variables local to a function. You

can ensure this by using a 'global'statement and initialising them in the part of the file where the figure is initialised.

We can access any property of any control using the 'Tag' property of the control and the MATLAB 'get()' and 'set()' functions.

value = get(handles.*ControlTagName*,'*PropertyName*');

set(handles.*ControlTagName*,'*PropertyName*','*Value*');

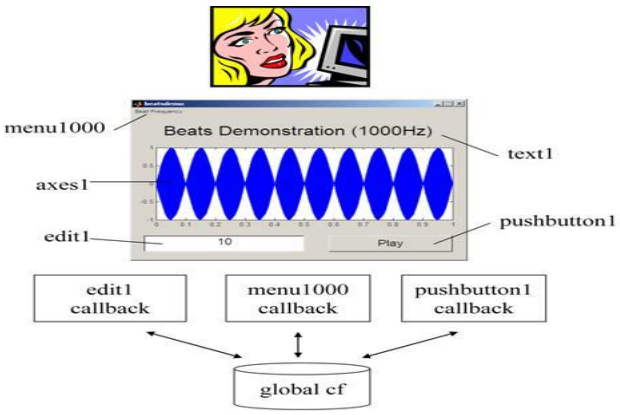
For example:

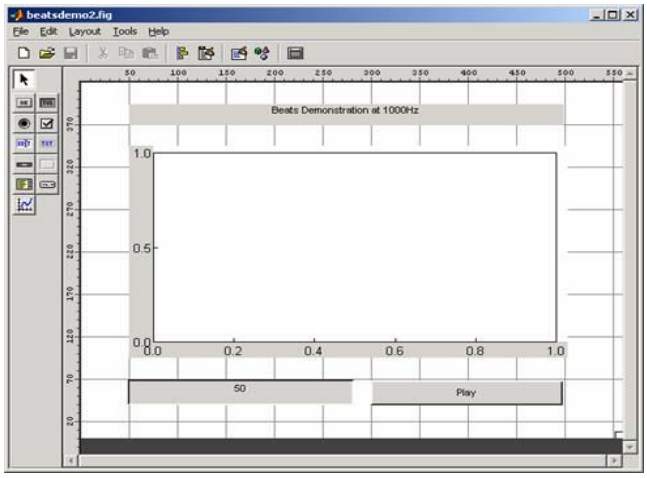
text = get(handles.edit1,'String');

set(handles.edit1,'String','100');

Note that most properties have to be get() and set() as strings. Use the num2str() and str2num() functions to help convert between strings and numeric values.

3. Worked example





GETTING INPUT, GETTING OUTPUT

uicontrol

Create user interface control object

Syntax

```
c = uicontrol
c = uicontrol(Name,Value,...)
c = uicontrol(parent)
c = uicontrol(parent,Name,Value,...)
uicontrol(c)
```

Description

c = uicontrol creates a uicontrol (push button) in the current figure and returns the uicontrol object, c. If there is no figure available, then MATLAB® creates a new figure to serve as the parent.

c = uicontrol(Name,Value,...) creates a uicontrol and specifies one or more uicontrol property names and corresponding values. Use this syntax to override the default uicontrol properties. The default uicontrol style is'pushbutton'.

c = uicontrol(parent) creates a uicontrol and designates a specific parent object. The parent argument can be a figure, uipanel, uibuttongroup, or uitab object.

c = uicontrol(parent,Name,Value,...) creates a uicontrol with a specific parent and one or more uicontrol properties.

uicontrol(c) gives focus to a specific uicontrol object, c.

Specifying the Uicontrol Style

- When selected, most uicontrol objects perform a predefined action. To create a specific type of uicontrol, set the Styleproperty as one of the following values. You can specify part of the Style value if it is unique among all the styles. For example, instead of 'radiobutton', you can specify 'radio'.
- 'checkbox' – A check box generates an action when you select it. Use check boxes to provide a number of independent choices. To activate a check box, click the mouse button on the object. The check box updates its appearance when its state changes.
- 'edit' – Editable text fields enable you to enter or modify text values. Use editable text when you want free text as input. To enable multiple lines of text, set Max-Min>1. Multiline edit boxes provide a vertical scroll bar for scrolling. The arrow keys also provide a way to scroll. Obtain the current text by getting the String property. The String property does not update as you type in an edit box. To execute the callback routine for an edit text control, type in the desired text and then do one of the following:
    - Click another component, the menu bar, or elsewhere on the window.
    - For a single line editable text box, press **Enter**.
    - For a multiline editable text box, press **Ctl+Enter**.
- 'frame'
- 'listbox' – List boxes display a list of items, from which you can select one or more items. Unlike pop-up menus, list boxes do not expand when clicked. The Min and Max properties control the selection mode:
    - To enable multiple selection of items, set Max-Min > 1.
    - To enable selection of only one item at a time, set Max-Min <= 1
- The Value property stores the row indexes of currently selected list box items, and is a vector value when you select multiple items. After any mouse button up event that changes the Value property, MATLAB evaluates the list box's callback routine. To delay action when multiple items can be selected, you can associate a "Done" push button with the list box. Use the callback for that button to evaluate the list box Value property.
- List boxes with the Enable property set to on differentiate between single and double left clicks. MATLAB sets the figure SelectionType property to normal or open accordingly before evaluating the list box Callback property. For enabled list boxes, **Ctrl**-left click and **Shift**-left click also set the figure SelectionType property to normal or open, respectively indicating a single or double click.
- 'popupmenu' – Pop-up menus (also known as drop-down menus) display a list of choices when you open them with a button-press. When closed, a pop-up menu indicates the current choice. Pop-up menus are useful when you want to provide a number of mutually exclusive choices, but do not want to take up the amount of space that a group of radio buttons requires.

- 'pushbutton' – Push buttons generate an action when activated. Left-click a push button to activate it. The button appears to depress until you release the mouse button. The callback activates when you release the mouse button while still pointing within the push button.
- 'radiobutton' – Radio buttons are similar to check boxes, but are intended to be mutually exclusive within a group of related radio buttons. When used this way, you can only select one radio button at any given time. To activate a radio button, click and release the mouse button over it. The easiest way to implement mutually exclusive behavior for a set of radio buttons is to place them within a uibuttongroup.
- 'slider' – Sliders accept numeric input within a specific range when you move the "thumb" button along a bar. The location of the thumb indicates a numeric value, assigned to the Value property when you release the mouse button. You can set the minimum, maximum, and current values, and step sizes of a slider.
- Move the thumb by doing any one of the following:
    - Press the mouse button on the thumb, and drag it along the bar.
    - Click in the bar or on arrow buttons located at both ends of the bar.
    - Click the keyboard arrow keys when the slider is in focus.
- 'text' – Static text boxes display lines of text. You typically use static text to label other controls, provide information to the user, or indicate values associated with a slider. If you assign the Callback property of a static text object to a function (or a character vector containing a MATLAB command), the static text will not respond when users try to interact with the text. However, you can code the Button DownFcn callback to respond to mouse clicks on the static text. 'togglebutton' – Toggle buttons are similar in appearance to push buttons, but they visually indicate their state, either 'on' (depressed) or 'off' (up). Clicking a toggle button changes its state, and switches its Value property between the toggle button's Min and Max values.
- ***Examples***

- Create uicontrols to allow users to adjust the appearance of a plot. For instance, create a program file called myui.m that contains the following code.

```
function myui
  % Create a figure and axes
  f = figure('Visible','off');
  ax = axes('Units','pixels');
  surf(peaks)

  % Create pop-up menu
  popup = uicontrol('Style', 'popup',...
       'String', {'parula','jet','hsv','hot','cool','gray'},...
       'Position', [20 340 100 50],...
       'Callback', @setmap);
```

```
    % Create push button
     btn = uicontrol('Style', 'pushbutton', 'String', 'Clear',...
        'Position', [20 20 50 20],...
        'Callback', 'cla');

    % Create slider
     sld = uicontrol('Style', 'slider',...
        'Min',1,'Max',50,'Value',41,...
        'Position', [400 20 120 20],...
        'Callback', @surfzlim);

     % Add a text uicontrol to label the slider.
     txt = uicontrol('Style','text',...
        'Position',[400 45 120 20],...
        'String','Vertical Exaggeration');

     % Make figure visble after adding all components
     f.Visible = 'on';
     % This code uses dot notation to set properties.
     % Dot notation runs in R2014b and later.
     % For R2014a and earlier: set(f,'Visible','on');

     function setmap(source,event)
        val = source.Value;
        maps = source.String;
        % For R2014a and earlier:
        % val = get(source,'Value');
        % maps = get(source,'String');

        newmap = maps{val};
        colormap(newmap);
     end

     function surfzlim(source,event)
        val = 51 - source.Value;
        % For R2014a and earlier:
        % val = 51 - get(source,'Value');

        zlim(ax,[-val val]);
     end
end
```

The resulting UI displays a plot. Users can adjust the color map, change the vertical scaling, or clear the axes.

PART-B(2 MARKS)

POSSIBLE QUESTIONS

1. What is Get and Set in MATLAB?
2. What is Manipulating a text?
3. List out some of the common toolboxes present in Matlab?
4. Write the syntax of while and for loop in MATLAB
5. What is randomizing a list?

PART-C(6 MARKS)

POSSIBLE QUESTIONS

1. Discuss about Manipulating a text in detail with example.
2. Explain about GUI in detail.
3. Explain about Writing a text to a file, reading from a file with example
4. Explain about Getting Input and Output in detail.
5. Explain about Randomizing and sorting a list with example.
6. Explain about attaching buttons to actions

KARPAGAM ACADEMY OF HIGHER EDUCATION

Department of Computer Science

II B.Sc(CS) (BATCH 2016-2018)

Programming In MATLAB

PART-A OBJECTIVE TYPE/ MULTIPLE CHOICE

	ONLINE EXAMINATIONS
S.No	**Question**
1	A program that response to event sis said to be _____
2	Graphical contraols and text boxes are created by the function _____
3	Toolbars are created by the fucntion
4	A _____ is a window on the computer screen
5	a most common container is a _____
6	_____ can contain components or other containers
7	A _____ is a graphical object that displays one or more text strings, which are specified in the text field's string property
8	a text field is created by _____
9	An _____ is a graphical object that allows a user to enter one or more text strings.
10	A _____ is a component that a user can click on to trigger a specific action
11	a _____ is a type of button that has two states on and off
12	_____ are graphical objects that display many lines of text and allow a user to select one or more of those lines
13	panels are created by the function _____
14	A _____ is a special type of figure that is sued to display information or to get input from a user
15	_____ may be modal or non modal
16	_____ boxes are typically used for warning and error messages
17	_____ boxes prompt a user to enter one or more values that may be used by a program
18	The _____ dialog boxes allows a user to interactively select a directory
19	If the user cancels the dialog box, _____ is set to zero
20	A _____ allows a user to select actions without additional components appearing on the GUI display
21	_____ menus are the pulled down from the menu bar at the top of a figure
22	_____ menus are pop up over the figure when a user right clicks the mouse over a graphical object
23	Accelerator keys are _____ combinations that cause a menu item to be executed without opening the menu first
24	_____ are single letters that can be presses to cause a menu item to execute once the menu is open
25	_____ create a generic dialog box
26	_____ function is used to create a standard menu, or amenu item on either a standard menu or a context menu
27	_____ is used to create a user defined toolbar
28	_____ is used to create a dialog box to ask a question

29	_____ is used to print the dialog box
30	_____ and keyboard mnemonics can be used to speed the operations of windows
31	The MATLAB graphics system is based on a hierarchical system of core _____
32	Each graphics object is known by aunique number called a _____
33	Each gaphics object has special data known as _____ associated with it
34	the highest level graphics object in MATLAB is the _____
35	Each _____ is a separetate window on the computer screen that can display graphical data
36	Each figure can contain _____ types of objects
37	The _____ is aunique integer or real number that is used by MATLAB to identify the object
38	Each property has a _____ and an associated value
39	When an _____ is created all of its poperties are automatically initialized to default values
40	The _____ is a just a variable which identifies the open file in your program.
41	The _____ function is the inverse of fprintf().
42	the function to get a line of text as characters
43	findstr() function is used to _____
44	GUI Stands for _____
45	_____ design system allows you to create applications consisting of one or more 'dialogs'
46	User can enter some text or numerical value by using _____
47	Program can display graphical result by using _____ control
48	We can edit the properties of the controls on the layout editor by right-clicking on them and choosing _____
49	We can add menu options to your dialog with the _____
50	You can control the 'shape' of the output matrix with a _____ argument.

: QUESTIONS

ONE MARK QUESTIONS

Option 1	Option 2	Option 3	Option 4	
program driven	event driven	events	none of the above	
figure	plot	uicontrol	control	
utool	uitoolbar	uimenu	uiaxes	
figure	container	plot	workspace	
figure	workspace	plot	area	
callbacks	panel	button group	component	
dynamic text field	text field	static text field	none of the above	
toolbox	uitoolbar	uicontrol	control box	
static text	tool box	edit boxes	menus	
pushbutton	tool box	static text field	menus	
pushbutton	tool box	static text box	toggle buttons	
toolbox	pushbutton	toggle button	list boxes	
unipanel	upanel	uipanel	panel	
toolbox	dialog boxes	toggle button	menus	
toolbox	dialog boxes	toggle button	menus	
non modal	modal	text boxes	list boxes	
output dialog	input dialog	text boxes	list boxes	
uiget	unisetdir	uigetdir	dirname	
directoryname	pathname	filename	figurename	
tools	list box	menus	dialog boxes	
context	standard	linear	collinear	
context	standard	linear	collinear	
CTRL + key	ALT + key	TAB+ key	DEL+Key	
Shortcut key	Keyboard mnemonics	Acclerator keys	none of the above	
arrdialog	create dialog	dialog	errdialog	
menus	create menu	uimenu	unicreate	
uimenu	unitools	toolbar	unitoolbar	
inputdlg	questdlg	question	dialog boxes	

inputdlg	printdlg	questdlg	errordlg	
Shortcut key	Keyboard mnemonics	Acclerator keys	none of the above	
graphics	system	graphics objects	properties	
handle	object	term	component	
object	properties	term	component	
directory	figures	root	path	
figure	plot	handle	object	
six	eight	two	seven	
handle	object	term	component	
term name	component name	property name	none of the above	
handle	term	object	data	
object	file handle	fgetl()	‘Menu Editor’	
fscanf()	fgetl()	object	‘Menu Editor’	
‘Menu Editor’	object	file handle	fgetl()	
Search a string	List a string	Compare a string	Delete a string	
Graphical User Input	Graphical User Interface	Geometric User Interface	Graphical Unique Interface	
GUI	object	file handle	‘Property Inspector’	
Menu options	Edit boxes	Figures	‘Property Inspector’	
Property Inspector	Figures	Edit boxes	GUI	
Property Inspector	Edit boxes	Figures	GUI	
Figures	Menu Editor	Edit boxes	GUI	
two	third	four	one	

	Answers
	event driven
	uicontrol
	uitoolbar
	container
	figure
	panel
	static text field
	uicontrol
	edit boxes
	pushbutton
	toggle buttons
	list boxes
	uipanel
	dialog boxes
	dialog boxes
	modal
	input dialog
	uigetdir
	directoryname
	menus
	standard
	context
	CTRL + key
	Keyboard mnemonics
	dialog
	uimenu
	unitoolbar
	questdlg

	printdlg
	Keyboard mnemonics
	graphics objects
	handle
	properties
	root
	figure
	seven
	handle
	property name
	object
	file handle
	fscanf()
	fgetl()
	Compare a string
	Graphical User Interface
	GUI
	Edit boxes
	Figures
	Property Inspector
	Menu Editor
	third

KARPAGAM ACADEMY OF HIGHER EDUCATION
(Deemed University Established Under Section 3 of UGC Act 1956)
COIMBATORE – 641 021

COMPUTER SCIENCE
Third Semester
FIRST INTERNAL EXAMINATION - July 2017

PROGRAMMING IN MATLAB

Class & Section: II B.Sc (CS) A&B **Duration: 2 hours**
Date & Session : **Maximum marks: 50 marks**
Subj.Code: 16CSU304B

PART- A (20 * 1= 20 Marks)
Answer ALL the Questions

1. Finite sequence of instructions is known as_____
   a)Program b)Flow Chart **c)Algorithm** d) Software
2. An algorithm expressed in a programming language is called as _____
   a)Expression **b)Computer program** c) Instruction d) Data
3. An algorithm can be expressed in a graphical form known as _____,
   a)Program b) Translator **c)Flow chart** d) Bar chart
4. A set of computer programs and related data that provide the instructions for telling computer hardware what to do and how to do it is _____
   a)Hardware **b)Software** c)Malware d)Shareware
5. Software designed to operate the computer hardware and to provide a platform for running _____
   a)System software b)Application software
   c)Operating Systems d)Utility programs
6. A collection of programs that form a bridge between user and the hardware is _____
   a)Operating systems b)Translators c)Software d)Program
7. Program that translates a set of code written in programming language into a machine code _____
   a)Translator b)Compiler c) Loader d) Linker
8. Which will translates assembly language programs into machine code ?
   a)Assembler b)Compiler c)interpreter d) editor
9. Compiler translates _____
   a) Object code into Machine Code b) Object code into Source Code
   c) Source code to assembly code **d) High level language code into object code**
10. What does Matlab stand for?
    a)Math Laboratory c) Mathworks
    b) Matrix Laboratory d) Nothing
11. The programs written in machine language are_____
    a) Machine independent **b) Machine dependent**
    c)Machine interconnected d) Machine interface
12. What symbol precedes all comments in Matlab?
    a) * c) //
    b) % d) <
13. Which of the following is not a pre-defined variable in Matlab.?

a) pi
b) inf
c) **i**
d) gravity

14. This Matlab command clears all data and variables stored in memory
    a) clc
    b) **clear**
    c) delete
    d) deallocate
15. Characters in Matlab are represented in their value in memory
    a) decimal
    b) ASCII
    c) **hex**
    d) string
16. Which of these is the way to access the first element in a vector named v (assuming there is at least one element in the vector)?
    a) v(0)
    b**) v(1)**
    c) v
    d) v(: , 0)
17. Which of the following is used to see if two elements are equal in MATLAB?
    a) !=
    b) ==
    c) **isequal**
    d) =
18. If vector = [1 2 3 4; 11 24 92 100; 345 65 90 1]. What will the value of a be equal to if this code is entered into MATLAB >>[a b] = size (vector)?
    a) 12
    c) 1
    d) 4
    e) **3**
19. What is the value of ans that is printed when the following code is run: isnumeric(32)
    a) **1**
    b) 0
    c) 32
    d) yes
20. To display Question 2 in command window, the correct command is
    a) disp(Question 2)
    b) display('Question 2')
    c) disp('Question 2')
    d) Question 2

PART- B (3 * 2= 6 Marks)
Answer ALL the Questions

21. List the Matrix operators.

Operator	**Purpose**	**Description**
-	Unary minus	-A negates the elements of A .
.*	Element-wise multiplication	A.*B is the element-by-element product of A and B .
.^	Element-wise power	A.^B is the matrix with elements A(i,j) to the B(i,j) power.
./	Right array division	A./B is the matrix with elements A(i,j)/B(i,j) .

22. **Mention type of operators used in MATLAB environment**.

MATLAB allows the following types of elementary operations −

- Arithmetic Operators
- Relational Operators

- Logical Operators
- Bitwise Operations
- Set Operations

23. Write in MATLAB 1. 32 + 5 2. 32+5

```
a= 1.32
b=5
c=2.35
d= (a+b)
e=(c+b)
disp(a)
disp(b)
```

PART C (3 * 8 = 24 Marks)
Answer ALL the Questions

24. a. Elaborate on Components of computers.

A computer system consists of mainly four basic units; namely input unit, storage unit, central processing unit and output unit. Central Processing unit further includes Arithmetic logic unit and control unit, as shown in the figure:. A computer performs five major operations or functions irrespective of its size and make. These are

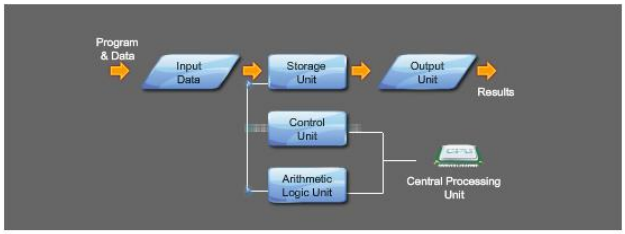
• it accepts data or instructions as input,

• it stores data and instruction

• it processes data as per the instructions,

• it controls all operations inside a computer, and

• it gives results in the form of output.



Functional Units:

a. Input Unit: This unit is used for entering data and programs into the computer system by the user for processing.

Basic Computer Organisation

b. Storage Unit: The storage unit is used for storing data and instructions before and after processing.

c. Output Unit: The output unit is used for storing the result as output produced by the computer after processing.

d. Processing: The task of performing operations like arithmetic and logical operations is called processing. The Central Processing Unit (CPU) takes data and instructions from the storage unit and makes all sorts of calculations based on the instructions given and the type of data provided. It is then sent back to the storage unit. CPU includes Arithmetic logic unit (ALU) and control unit (CU)

Arithmetic Logic Unit: All calculations and comparisons, based on the instructions provided, are carried out within the ALU. It performs arithmetic functions like addition, subtraction, multiplication, division and also logical operations like greater than, less than and equal to etc.

• Control Unit: Controlling of all operations like input, processing and output are performed by control unit. It takes care of step by step processing of all operations in side the computer.

Memory

Computer's memory can be classified into two types; primary memory and secondary memory

RAM

a. Primary Memory can be further classified as **RAM and ROM**.

• RAM or Random Access Memory is the unit in a computer system. It is the place in a computer where the operating system, application programs and the data in current use are kept temporarily so that they can be accessed by the computer's processor. It is said to be 'volatile' since its contents are accessible only as long as the computer is on. The contents of RAM are no more available once the computer is turned off.

ROM or Read Only Memory is a special type of memory which can only be read and contents of which are not lost even when the computer is switched off. It typically contains manufacturer's instructions. Among other things, ROM also stores an initial program called the 'bootstrap loader' whose function is to start the operation of computer system once the power is turned on.

b. Secondary Memory

RAM is volatile memory having a limited storage capacity. Secondary/auxiliary memory is storage other than the RAM. These include devices that are peripheral and are connected and controlled by the computer to enable permanent storage of programs and data.

- CD ROM

Secondary storage devices are of two types; magnetic and optical. Magnetic devices include hard disks and optical storage devices are CDs, DVDs, Pen drive, Zip drive etc.

• Hard Disk

Hard disks are made up of rigid material and are usually a stack of metal disks sealed in a box. The hard disk and the hard disk drive exist together as a unit and is a permanent part of the computer where data and programs are saved. These disks have storage capacities ranging from 1GB to 80 GB and more. Hard disks are rewritable.

• Compact Disk

Compact Disk (CD) is portable disk having data storage capacity between 650-700 MB. It can hold large amount of information such as music, full-motion videos, and text etc. CDs can be either read only or read write type.

CD Drive

• Digital Video Disk

Digital Video Disk (DVD) is similar to a CD but has larger storage capacity and enormous clarity. Depending upon the disk type it can store several Gigabytes of data. DVDs are primarily used to store music or movies and can be played back on your television or the computer too. These are not rewritable.

Hard Disk

Input / Output Devices:

These devices are used to enter information and instructions into a computer for storage or processing and to deliver the processed data to a user. Input/Output devices are required for users to communicate with the computer. In simple terms, input devices bring information INTO the computer and output devices bring information OUT of a computer system. These input/output devices are also known as peripherals since they surround the CPU and memory of a computer system.

Input Devices

An input device is any device that provides input to a computer. There are many input devices, but the two most common ones are a keyboard and mouse. Every key you press on the keyboard and every movement or click you make with the mouse sends a specific input signal to the computer.

Keyboard

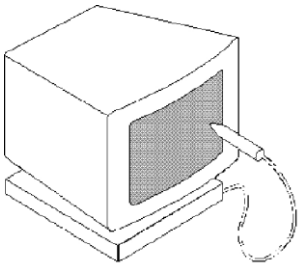
• **Keyboard**: The keyboard is very much like a standard typewriter keyboard with a few additional keys. The basic QWERTY layout of characters is maintained to make it easy to use the system. The additional keys are included to perform certain special functions. These are known as function keys that vary in number from keyboard to keyboard.

• **Mouse**: A device that controls the movement of the cursor or pointer on a display screen. A mouse is a small object you can roll along a hard and flat surface. Its name is derived from its shape, which looks a bit like a mouse. As you move the mouse, the pointer on the display screen moves in the same direction.

• **Trackball**: A trackball is an input device used to enter motion data into computers or other electronic devices. It serves the same purpose as a mouse, but is designed with a moveable ball on the top, which can be rolled in any direction.

• **Touchpad**: A touch pad is a device for pointing (controlling input positioning) on a computer display screen. It is an alternative to the mouse. Originally incorporated in laptop computers, touch pads are also being made for use with desktop computers. A touch pad works by sensing the user's finger movement and downward pressure. • Touch Screen: It allows the user to operate/make selections by simply touching the display screen. A display screen that is sensitive to the touch of a finger or stylus. Widely used on ATM machines, retail point-of-sale terminals, car navigation systems, medical monitors and industrial control panels.

Light Pen: Light pen is an input device that utilizes a light-sensitive detector to select objects on a display screen.



• **Magnetic ink character recognition (MICR)**: MICR can identify character printed with a special ink that contains particles of magnetic material. This device particularly finds applications in banking industry.

• **Optical mark recognition (OMR)**: Optical mark recognition, also called mark sense reader is a technology where an OMR device senses the presence or absence of a mark, such as pencil mark. OMR is widely used in tests such as aptitude test.

• **Bar code reader**: Bar-code readers are photoelectric scanners that read the bar codes or vertical zebra strips marks, printed on product containers. These devices are generally used in super markets, bookshops etc.

Scanner

Scanner is an input device that can read text or illustration printed on paper and translates the information into a form that the computer can use. A scanner works by digitizing an image.

Output Devices:

Output device receives information from the CPU and presents it to the user in the desired from. The processed data, stored in the memory of the computer is sent to the output unit, which then converts it into a form that can be understood by the user. The output is usually produced in one of the two ways – on the display device, or on paper (hard copy).

•**Monitor**: is often used synonymously with "computer screen" or "display." Monitor is an output device that resembles the television screen (fig. 1.8). It may use a Cathode Ray

Tube (CRT) to display information. The monitor is associated with a keyboard for manual input of characters and displays the information as it is keyed in. It also displays the program or application output. Like the television, monitors are also available in different sizes. • **Printer**: Printers are used to produce paper (commonly known as hard copy) output. Based on the technology used, they can be classified as Impact or Non-impact printers.

Impact printers use the typewriting printing mechanism wherein a hammer strikes the paper through a ribbon in order to produce output. Dot-matrix and Character printers fall under this category.

Non-impact printers do not touch the paper while printing. They use chemical, heat or electrical signals to etch the symbols on paper. Inkjet, Deskjet, Laser, Thermal printers fall under this category of printers.

Plotter: Plotters are used to print graphical output on paper. It interprets computer commands and makes line drawings on paper using multi colored automated pens. It is capable of producing graphs, drawings, charts, maps etc. • **Facsimile (FAX)**: Facsimile machine, a device that can send or receive pictures and text over a telephone line. Fax machines work by digitizing an image.

Sound cards and Speaker(s): An expansion board that enables a computer to manipulate and output sounds. Sound cards are necessary for nearly all CD-ROMs and have become commonplace on modern personal computers. Sound cards enable the computer to output sound through speakers connected to the board, to record sound input from a microphone connected to the computer, and manipulate sound stored on a disk.

(OR)

b. Discuss on software hierarchy.

Software hierarchy

The lowest level description of a computer program is just the sequence of numbers which encode the basic CPU operations. This level is called **machine code**. Machine code is specific to a given CPU manufacturer and often specific to a given model type (for example the Pentium CPU has some codes not used by earlier 8086 CPUs). Machine code is very difficult for a human to read or write, so the lowest level of programming done by humans is in a language in which each basic operation is given a mnemonic code called **assembly language**. Humans can read and write using assembly language which can be converted into machine code using an **assembler**. Assembly language, like machine code is often specific to a particular CPU manufacturer or model.

The development of **high-level languages** meant that humans could program using a formalism that was closer to their conceptual models of the data being manipulated: characters, real numbers, lists, tables or database records. Such languages are easier for humans to learn and to use, and furthermore they tend to be available across different computers; with each manufacturer supplying a conversion program between the high-level language and the assembly language for their CPU. Examples of high-level languages are Fortran, Pascal, Basic, C, C++, Java and MATLAB.

Modern computer systems need to deal with complex tasks involving multiple programs interacting simultaneously, and the sharing of access to files on disks, to network resources and displays. To cope with these demands, manufacturers supply **operating systems** (e.g. Windows, Linux), which are themselves programs which help the user operate the computer and run other **application** programs. Often individual application programs need to work together to achieve an objective: for example a word processing application might call on a drawing package or on a spreadsheet program to do some specific processing within a document. This idea of combining programs is called **scripting**, where the specifications for which programs are to be executed and how they should interact is specified in a **script**.

25. a. Explain various Matlab windows with example.

Programming Environment

MATLAB WINDOWS

It is assumed that the software is installed on the computer, and that the user can start the program. Once the program starts, the MATLAB desktop window opens (Figure 1-1). The window contains four smaller windows: the Command Window, the Current Folder Window, the Workspace Window, and the Command History Window. This is the default view that shows four of the various windows of MATLAB. A list of several windows and their purpose is given in Table 1-1. The Start button on the lower left side can be used to access MATLAB tools and features. Four of the windows—the Command Window, the Figure Window, the Editor Window, and the Help Window—are used extensively throughout the book and are briefly described on the following pages

Command Window: The Command Window is MATLAB's main window and opens when MATLAB is started. It is convenient to have the Command Window as the only visible window, and this can be done by either closing all the other windows (click on the x at the top right-hand side of the window you want to close) or by first selecting the Desktop Layout in the Desktop menu, and then 6 Chapter 1: Starting with MATLAB selecting Command Window Only from the submenu that opens

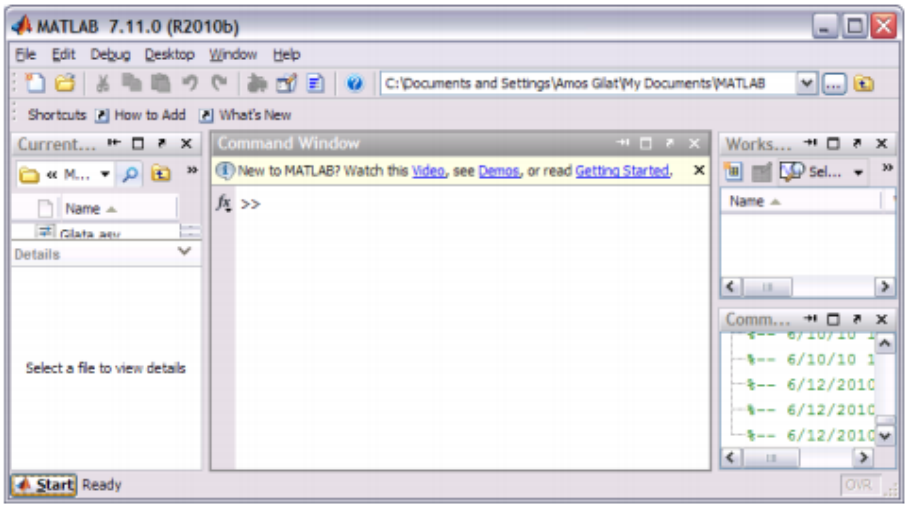


Figure 1-1: The default view of MATLAB desktop.

Table 1-1: MATLAB windows

Window	Purpose
Command Window	Main window, enters variables, runs programs.
Figure Window	Contains output from graphic commands.
Editor Window	Creates and debugs script and function files.
Help Window	Provides help information.
Command History Window	Logs commands entered in the Command Window.
Workspace Window	Provides information about the variables that are used.
Current Folder Window	Shows the files in the current folder.

Figure Window: The Figure Window opens automatically when graphics commands are executed, and contains graphs created by these commands. An example of a Figure Window is shown in Figure 1-2.

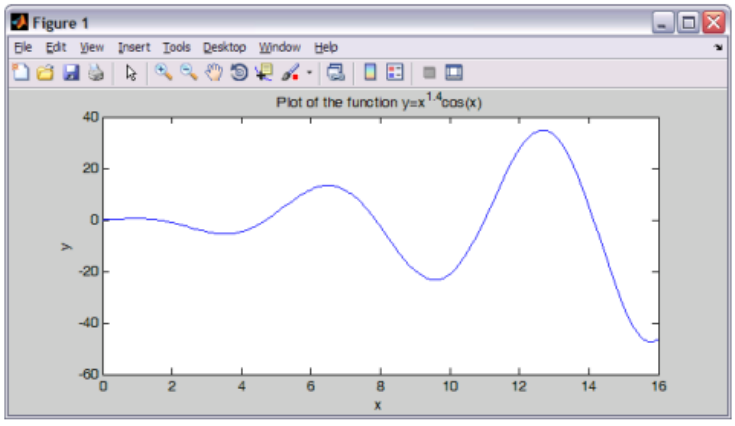


Figure 1-2: Example of a Figure Window.

Editor Window: The Editor Window is used for writing and editing programs. This window is opened from the File menu. An example of an Editor Window is shown in Figure 1-3.

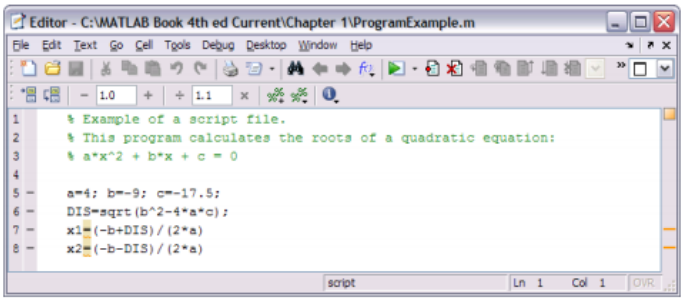


Figure 1-3: Example of an Editor Window.

Help Window: The Help Window contains help information. This window can be opened from the Help menu in the toolbar of any MATLAB window. The Help Window is interactive and can be used to obtain information on any feature of MATLAB. Figure 1-4 shows an open Help Window.

Figure 1-4: The Help Window.

Working In The Command Window The Command Window is MATLAB's main window and can be used for executing commands, opening other windows, running programs written by the user, and managing the software. An example of the Command Window, with several simple commands that will be explained later in this chapter, is shown in Figure 1-5.

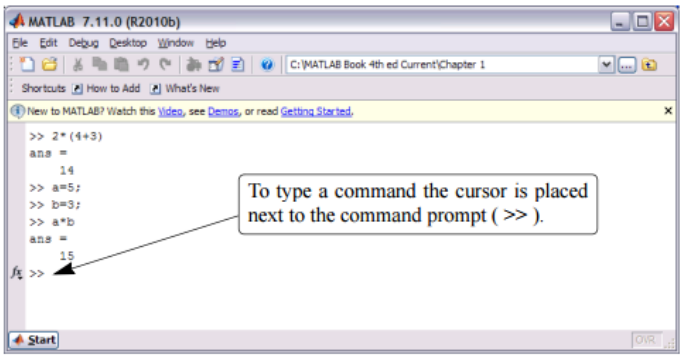


Figure 1-5: The Command Window.

A FIRST PROGRAM

Matlab stores most of its numerical results as matrices. Unlike some languages (C, C++, C#), it dynamically allocates memory to store variables. Therefore, it is not necessary to declare variables before using them. Let's begin by simply adding two numbers. Click in the Command Window. You will see a flashing "|" symbols next to the ">>" symbol. Enter the following commands

1. Type in "x = 3" then hit "enter"
2. Type in "y = 2;" then hit "enter" (note the semicolon here!)
3. Type "z = x + y" then hit "enter"

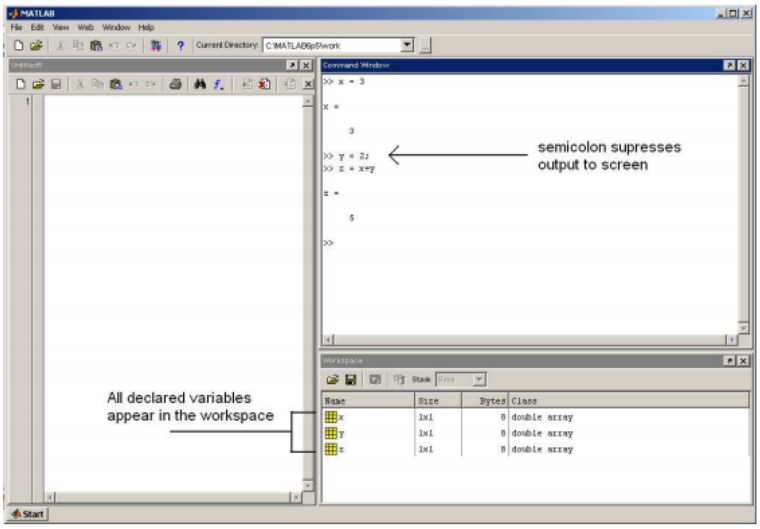


Figure 4: Entering in scalar values into **Matlab**

All declared variables appear in the workspace. Recall that these values are stored as matrices. The "size" column tells us the dimension of the matrix. As expected, all these variables are 1x1 scalar values. To double check on value stored in this matrix, simply double click any of the variables in the Workspace.

Example program

The command

disp(*argument*);

displays the value of the argument. This can be a number, a string in single quotes, or an expression. For simple numbers, the arithmetic operators are: **+, -, *,** / and**^.** Try

```
disp(2*3+1);
```

or

```
disp('Hello World!');
```

Try these programs out first on the command line; then practise using the editor to enter the commands, saving them to a file, loading the file and running the program from inside the editor.

(OR)

b. Write note on different Array operators with example in Matlab.

Arrays

MATLAB is particularly powerful in the way it deals with tables of data, called arrays. An array is simply a variable that can contain a number of values arranged in tabular form. Arrays may be one dimensional (like a list), two dimensional (like a table), or have more dimensions. To set the value of one element of a one dimensional array, use the notation

variable(*index*)=*expression*;

for example

```
table(1)=3;
table(2)=6;
```

Note that indexes must be expressions evaluating to positive integers. The smallest index is 1. To access one element from a one dimensional array, use the notation

variable(*index*)

for example

```
a=table(2);
disp(table(2));
```

For two dimensional arrays, use

variable(*index*,*index*)=*expression*;

to set the value and

variable(*index*,*index*)

to retrieve its value. You can store strings in tables, but each string occupies a row, and all rows must be the same length (think of a two-dimensional array of characters).

You can assign a whole array in one operation using a notation involving square brackets: for example:

array = [v11 v12 v13; v21 v22 v23];

where v11 is the value in row 1 col 1; v21 is the value in row 2 col 1; etc. The ';' marks the end of a row.

You can generate arrays containing sequences very easily with the ':' operator. The expression

start:*stop*

generates a sequence of integers from start to stop. The expression

start:*increment*:*stop*

generates a sequence from start to stop with the specfied increment. Try

```
disp(1:10);
disp(1:2:10);
```

You can also select sub-parts of the array with the ':' operator. For example,

```
x(3:5)
```

represents the array consisting of the third through fifth elements of x. Also

```
y(2:2:100)
```

represents the array containing the even number elements of y below index 100.

You can also add subtract, multiply and divide arrays of data using the operators we've mentioned previously. However MATLAB makes a difference between operations that work on a cell-by-cell basis (so-called "dot" operations) as opposed to operations that work on the arrays as a whole. For example, if you want to multiply two arrays of equal size to give a third array in which each cell contains the product of the corresponding cells in the input, then you need to use the "dot-multiply" operator .* for example

```
C = A.*B;
```

Finally you can transpose rows and columns of a matrix with the ' operator, for example

```
disp(A')
```

26. a. Explain Variables and assignment in Matlab.

Variables and assignment

Variables are named locations in memory where numbers, strings and other elements of data may be stored while the program is working. Variable names are combinations of letters and digits, but must start with a latter. MATLAB does not require you to declare the names of variables in advance of their use. This is actually a common cause of error, since it allows you to refer accidentally to variables that don't exist. To assign a variable a value, use the **assignment statement**. This takes the form

variable=expression;

for example

```
a=6;
```

or

name='Mark';

To display the contents of a variable, use

disp(*variable*);

Please note that −

- Once a variable is entered into the system, you can refer to it later.
- Variables must have values before they are used.
- When an expression returns a result that is not assigned to any variable, the system assigns it to a variable named ans, which can be used later.

For example,

```
sqrt(78)
```

MATLAB will execute the above statement and return the following result −

```
ans =  8.8318
```

You can use this variable **ans** −

```
sqrt(78);
9876/ans
```

MATLAB will execute the above statement and return the following result −

```
ans =  1118.2
```

Let's look at another example −

```
x = 7 * 8;
y = x * 7.89
```

MATLAB will execute the above statement and return the following result −

```
y =  441.84
```

Multiple Assignments

You can have multiple assignments on the same line. For example,

```
a = 2; b = 7; c = a * b
```

MATLAB will execute the above statement and return the following result −

```
c = 14
```

(OR)

27. b. Set up a vector called N with five elements having the values: 1, 2, 3, 4, 5. Using N,

create assignment statements for a vector X which will result in X having these values:

a. 2, 4, 6, 8, 10 b. 1/2, 1, 3/2, 2, 5/2 c. 1, 1/2, 1/3, 1/4, 1/5 d. 1, 1/4, 1/9, 1/16, 1/25

```
N=[1 2 3 4 5];
X=2*N;
disp(X);

X=N/2;
disp(X);

X=1./N;
disp(X);

X=1./(N.*N);
disp(X);
```

a.

Register Number_____

[16CSU304B]

KARPAGAM ACADEMY FOR HIGHER EDUCATION

KARPAGAM UNIVERSITY

(Deemed University Established Under Section 3 of UGC Act 1956)

Coimbatore - 641021.

(For the candidates admitted from 2016 onwards)

SECOND INTERNAL EXAMINATION, AUGUST 2017

Third Semester

COMPUTER SCIENCE

PROGRAMMING IN MATLAB

Maximum : 50 Marks **Duration: 2 Hours**

PART-A (20 X 1 = 20 Marks)

(Answer ALL the Questions)

1. In MATLAB the matrix is defined as an _____

a)vector b)scalar **c)array** d)integer

2. The fundamental unit of data in any MATLAB program is the _____

a)array b)vector c)scalar d)none

3. the operator, == stands for _____

a)not equal to **b)equal to** c)assigned to d)approximately equal to

4. The _____ function accepts an array argument and displays the value of the array in the command window

a)disp b)format c)special d)fprintf

5._____ are operations performed between arrays on an element by element basis

a)matrix operations c)vector operations

b)array operations d)arithmetic operations

6.To add a comment to the mfile, the MATLAB command is

a)% b) ; c)comment(' ') d)&

7.To add a superscript, use the character(s)

a)\^ **b) ^** c)\super d) \s

8.The standard inputs for the loglog command are

a)(log(x), y) **b) (x,y)** c) (log(x),log(y)) d) (log10(x),log10(y))

9. When a _____ command is used the additional plots will be laid on top of the previously existing plots

a) hold on b) hold off c) holded on d)none

10. The MATLAB command to make a plot is

a) figure b) fit **c)plot** d) pplot

11. The _____ command returns the number of elements of the matrix in each dimensions
    a) len(x) b)length(x) **c)size(x)** d)none
12. A MATLAB function is a special type of _____ that runs in its own independent workspace
    a) G file **b)M file** c)MM file d)MX file

13. _____ function determines if the first n characters of two strings are identical
    a) strncmp b)strcmp c)strcmpi d)stricmp
14. Variable can be converte dfrom double data type to char data type using _____ function
    a) char b)int c)double d)string
15. In _____ function the output goes into a character string instead of the command window
    a)fprintf **b)sprintf** c)printf d)print
16. _____ function replaces onestring with another
    a)strfind b)strmatch **c)strrep** d)strrrev
17. The _____ command sets the axis increments to be equal on both axes
    a)axis normal b)axis square c)axis on **d)axis equal**
18. _____ function is used to justify the string
    a)strjust b)strmatch c)strrep d)strrrev
19. _____ are just collections of MATLAB statements that are stored in a file
    a)function files **b)script files** c)legal files d)none
20. When used in the fprintf command, the %g is used as the
    a)single character display c)string notation display
    b)fixed point display **d)default number display**

PART-B (3 X 2 = 6 Marks)
(Answer ALL the Questions)

21. What is an Array?

MATLAB is an abbreviation for "matrix laboratory." While other programming languages mostly work with numbers one at a time, MATLAB® is designed to operate primarily on whole matrices and arrays.

All MATLAB variables are multidimensional arrays, no matter what type of data. A matrix is a two-dimensional array often used for linear algebra.

Array Creation

To create an array with four elements in a single row, separate the elements with either a comma (,) or a space.

```
a = [1 2 3 4]
a =
```

1 2 3 4

22. What is an M-File?

An m-file, or script file, is a simple text file where we can place MATLAB commands. When the file is run, MATLAB reads the commands and executes them exactly as it would if we had typed each command sequentially at the MATLAB prompt.

23. What is graph plots?

plot(X,Y) creates a 2-D line plot of the data in Y versus the corresponding values in X.

- If X and Y are both vectors, then they must have equal length. The plot function plots Y versus X.
- If X and Y are both matrices, then they must have equal size. The plot function plots columns of Yversus columns of X.

PART-C (3 X 8 = 24 Marks)
(Answer ALL the Questions)

24. a) Explain in detail about array and its types with suitable example.

Array:

MATLAB is particularly powerful in the way it deals with tables of data, called arrays. An array is simply a variable that can contain a number of values arranged in tabular form. Arrays may be one dimensional (like a list), two dimensional (like a table), or have more dimensions. To set the value of one element of a one dimensional array, use the notation

variable(*index*)=*expression*;

for example

```
table(1)=3;
table(2)=6;
```

Note that indexes must be expressions evaluating to positive integers. The smallest index is 1. To access one element from a one dimensional array, use the notation

variable(*index*)

for example

```
a=table(2);
disp(table(2));
```

For two dimensional arrays, use

variable(*index,index*)=*expression*;

to set the value and

variable(*index*,*index*)

to retrieve its value. You can store strings in tables, but each string occupies a row, and all rows must be the same length (think of a two-dimensional array of characters).

You can assign a whole array in one operation using a notation involving square brackets: for example:

array = [v11 v12 v13; v21 v22 v23];

where v11 is the value in row 1 col 1; v21 is the value in row 2 col 1; etc. The ';' marks the end of a row.

You can generate arrays containing sequences very easily with the ':' operator. The expression

start:*stop*

generates a sequence of integers from start to stop. The expression

start:*increment*:*stop*

generates a sequence from start to stop with the specfied increment. Try

```
disp(1:10);
disp(1:2:10);
```

You can also select sub-parts of the array with the ':' operator. For example,

x(3:5)

represents the array consisting of the third through fifth elements of x. Also

y(2:2:100)

represents the array containing the even number elements of y below index 100.

You can also add subtract, multiply and divide arrays of data using the operators we've mentioned previously. However MATLAB makes a difference between operations that work on a cell-by-cell basis (so-called "dot" operations) as opposed to operations that work on the arrays as a whole. For example, if you want to multiply two arrays of equal size to give a third array in which each cell contains the product of the corresponding cells in the input, then you need to use the "dot-multiply" operator .* for example

```
C = A.*B;
```

Finally you can transpose rows and columns of a matrix with the ' operator, for example

```
disp(A')
```

[OR]

b) Explain in detail about variables and assignment statements.

Variables are named locations in memory where numbers, strings and other elements of data may be stored while the program is working. Variable names are combinations of letters and digits, but must start with a latter.

MATLAB does not require you to declare the names of variables in advance of their use. This is actually a common cause of error, since it allows you to refer accidentally to variables that don't exist. To assign a variable a value, use the **assignment statement**. This takes the form

variable=expression;

for example

a=6;

or

name='Mark';

To display the contents of a variable, use

disp(*variable*);

- Once a variable is entered into the system, you can refer to it later.
- Variables must have values before they are used.
- When an expression returns a result that is not assigned to any variable, the system assigns it to a variable named ans, which can be used later.

For example,

```
sqrt(78)
```

MATLAB will execute the above statement and return the following result −

```
ans =  8.8318
```

You can use this variable **ans** −

```
sqrt(78);
```

```
9876/ans
```

MATLAB will execute the above statement and return the following result −

```
ans =  1118.2
```

Let's look at another example −

```
x = 7 * 8;
y = x * 7.89
```

MATLAB will execute the above statement and return the following result −

```
y =  441.84
```

Multiple Assignments

You can have multiple assignments on the same line. For example,

```
a = 2; b = 7; c = a * b
```

MATLAB will execute the above statement and return the following result −

```
c = 14
```

25. a) Explain about Basic Plotting in detail.

Basic Plotting

To create XY graphs, it is easiest to form your data into two row vectors, one for the x co-ordinates, and one for the y co-ordinates. The command

plot(x,y)

will then create a figure with points at each y value for each matching x value. You can control the style of any line drawn through the points by a third string argument to the plot command:

plot(x,y,style);

where style is made up from characters as follows:

- Color strings are 'c', 'm', 'y', 'r', 'g', 'b', 'w', and 'k'. These correspond to cyan, magenta, yellow, red, green, blue, white, and black.
- Linestyle strings are '-' for solid, '--' for dashed, ':' for dotted, '-.' for dash-dot, and none for no line.

The marker types are '+', 'o', '*', and 'x' and the filled marker types 's' for square, 'd' for diamond, '^' for up triangle, 'v' for down triangle, '>' for right triangle, '<' for left triangle, 'p' for pentagram, 'h' for hexagram, and none for no marker.

For example:

```
x = [ 1 2 3 4 ];
y = [ 10 15 20 25 ];
plot(x,y,'g-*');
```

You can plot multiple lines by repeating the arguments:

```
plot(x1,y1,x2,y2,…);
```

or

```
plot(x1,y1,style1,x2,y2,style2,…);
```

You can give the graph a title with the

```
title(label);
```

command, where label is a character string. Likewise you can add labels to the X and Y axes with

```
xlabel(label);
```

and

```
ylabel(label);
```

You can add a legend with

```
legend(label1,label2,label3,…);
```

Description

Plotting functions accept line specifications as arguments and modify the graph generated accordingly. You can specify these three components:

- Line style
- Marker symbol
- Color

Line Style Specifiers

You indicate the line styles, marker types, and colors you want to display, detailed in the following tables:

Specifier	LineStyle
'-'	Solid line (default)
'--'	Dashed line
':'	Dotted line
'-.'	Dash-dot line

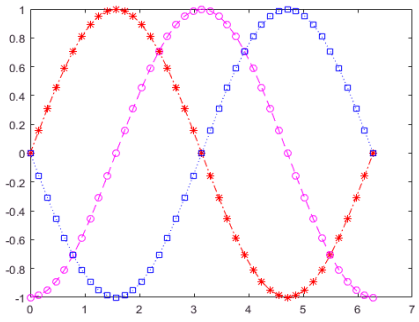
Marker Specifiers

Specifier	Marker Type
'+'	Plus sign
'o'	Circle
'*'	Asterisk
'.'	Point
'x'	Cross
'square' or 's'	Square
'diamond' or 'd'	Diamond
'^'	Upward-pointing triangle
'v'	Downward-pointing triangle
'>'	Right-pointing triangle
'<'	Left-pointing triangle
'pentagram' or 'p'	Five-pointed star (pentagram)
'hexagram' or 'h'	Six-pointed star (hexagram)

Color Specifier

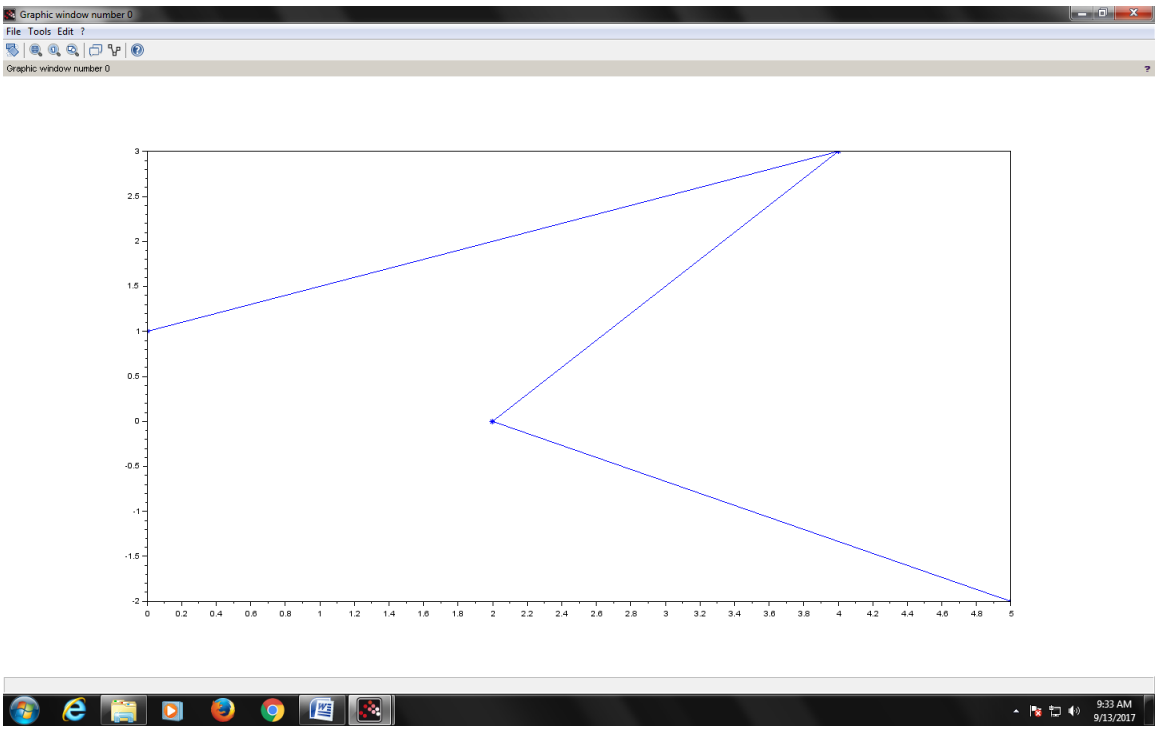
Specifier	Color
r	Red
g	Green
b	Blue
c	Cyan
m	Magenta
y	Yellow
k	Black
w	White

```
figure

t = 0:pi/20:2*pi;

plot(t,sin(t),'-.r*')

hold on

plot(t,sin(t-pi/2),'--mo')

plot(t,sin(t-pi),':bs')

hold off
```



Example Program

```
X=[0 4 2 5];
Y=[1 3 0 -2];
plot(X,Y,'*-');
```



[OR]

b) Explain in detail about M-Files.

M-FILES

MATLAB allows writing two kinds of program files −

- **Scripts** − script files are program files with **.m extension**. In these files, you write series of commands, which you want to execute together. Scripts do not accept inputs and do not return any outputs. They operate on data in the workspace.

- **Functions** − functions files are also program files with **.m extension**. Functions can accept inputs and return outputs. Internal variables are local to the function.

You can use the MATLAB editor or any other text editor to create your **.m**files. In this section, we will discuss the script files. A script file contains multiple sequential lines of MATLAB commands and function calls. You can run a script by typing its name at the command line.

Creating and Running Script File

To create scripts files, you need to use a text editor. You can open the MATLAB editor in two ways:

- Using the command prompt
- Using the IDE

If you are using the command prompt, type **edit** in the command prompt. This will open the editor. You can directly type **edit** and then the filename (with .m extension)

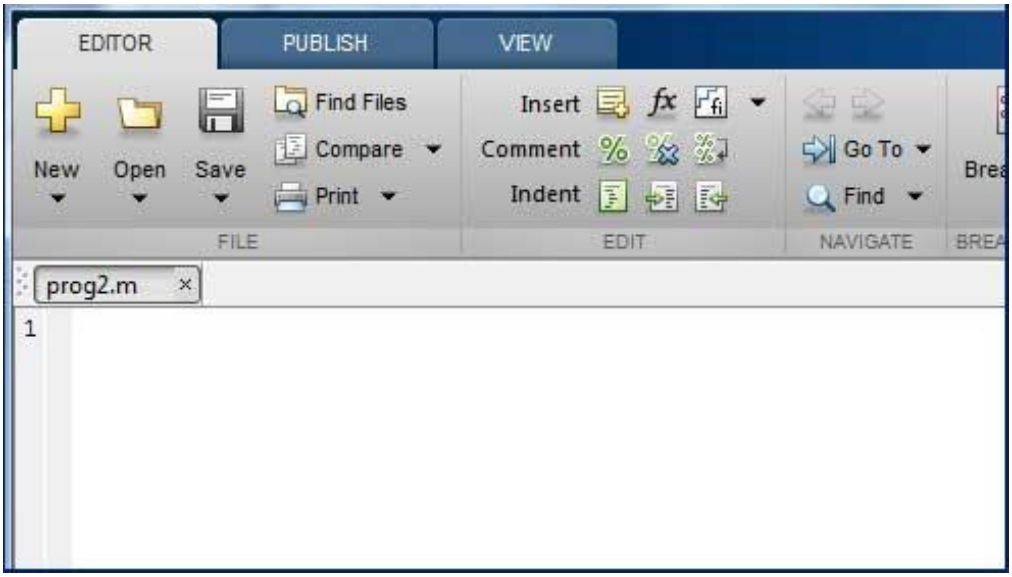
```
edit
Or
edit <filename>
```

The above command will create the file in default MATLAB directory. If you want to store all program files in a specific folder, then you will have to provide the entire path.

Let us create a folder named progs. Type the following commands at the command prompt (>>):

```
mkdir progs    % create directory progs under default directory
chdir progs    % changing the current directory to progs
edit  prog1.m  % creating an m file named prog1.m
```

If you are creating the file for first time, MATLAB prompts you to confirm it. Click Yes.



Alternatively, if you are using the IDE, choose NEW -> Script. This also opens the editor and creates a file named Untitled. You can name and save the file after typing the code.

Type the following code in the editor −

```
NoOfStudents = 6000;
TeachingStaff = 150;
NonTeachingStaff = 20;
Total = NoOfStudents + TeachingStaff ...
  + NonTeachingStaff;
disp(Total);
```

After creating and saving the file, you can run it in two ways −

- Clicking the **Run** button on the editor window or
- Just typing the filename (without extension) in the command prompt: >> prog1

The command window prompt displays the result −

```
6170
```

Example

Create a script file, and type the following code −

```
a = 5; b = 7;
c = a + b
d = c + sin(b)
e = 5 * d
f = exp(-d)
```

When the above code is compiled and executed, it produces the following result −

```
c =  12
d =  12.657
e =  63.285
f =   3.1852e-06
```

26. a) Explain about Generating wave forms, Sound, replay, load and save in detail.

GENERATING WAVEFORMS

Waveforms are just long vectors with one number per amplitude sample. Usually they are best kept scaled so that each amplitude is between –1 and 1. To generate a sinewave, first generate a time sequence t representing the times of each sampling instant; for example:

```
t = 0:0.0001:2;
```

would generate a two second sequence with a sampling interval of 0.1ms (i.e. 10,000Hz). You can then generate a sinewave at frequency F with the expression

```
y = sin(2*pi*F*t);
```

You can create a pulse by creating a vector of zeros and setting a single element to one. A pulse train has a series of elements set to one. If these occurred every 100 elements, you might use the expression

```
y(1:100:10000)=1;
```

To create a simple sawtooth, you can use the remainder function, for example

```
y = rem(1:10000,100)/100;
```

To create a noise waveform, you can use the 'rand(*nrows,ncols*)' function, for example

```
y = rand(1,10000);
```

SOUND REPLAY, LOAD AND SAVE

To replay a waveform, you can use

sound(*wave,samplerate*);

To ensure that the waveform is scaled to the range –1 .. +1 before replay, use

soundsc(*wave,samplerate*);

instead.

To save a waveform to a file, use

save *filename variable*;

To load a waveform from a file, use

load *filename variable*;

To save a waveform in a Windows compatible audio file format, use

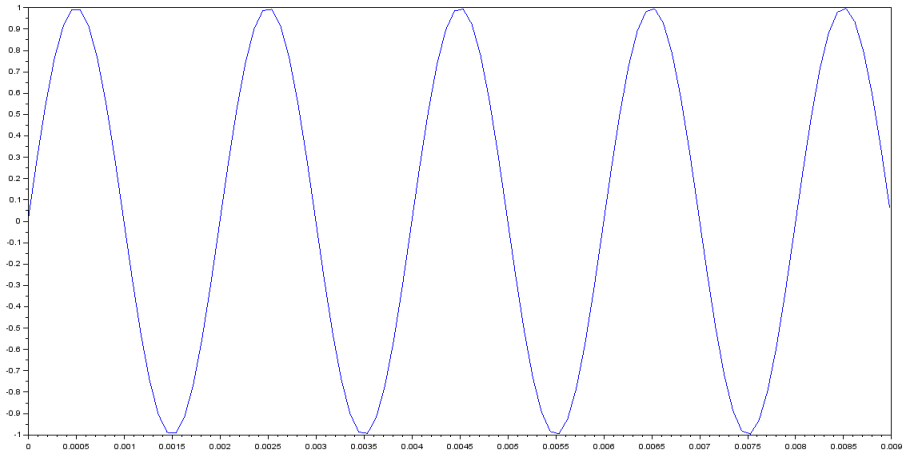
wavwrite(*waveform,samplerate,filename*);

To load a Windows compatible audio file, use

[*waveform,samplerate,nbits*]=wavread(*filename*);

Example program:

```
srate=11025;
t=0:1/srate:1;
s=sin(2*%pi*500*t);
sound(s,srate);
wavwrite(s,srate,16,'ex35.wav');
plot(t(1:100),s(1:100),'-');
```



[OR]

b) Explain in detail about Formatted Console Input- Output.

We can control the exact way in which values are printed to the screen with the 'fprintf()' function (fprintf= "file print formatted"). This function takes one argument repesenting the formatting instructions, followed by a list of values to be printed. Embedded within the format string are 'percent commands' which control where and how the values are to be written. Here are some examples:

```
fprintf('The answer is %g seconds.\n',nsec);
fprintf('Day of the week = %s\n',dayofweek([7 12 1941]));
fprintf('Mean=%.3f ± %.4f\n',mean,stddev);
```

The command %g represents a general real number, %f means a fixed point number, %d a decimal integer, and %s a string. You can put numeric values between the '%' and the letter to control the field width and the number of digits after the decimal point. For example (□=space):

fprintf('%5g',10)	□□□10
fprintf('%10.4f',123.456)	□□123.4560
fprintf('%10s', 'fred')	□□□□□□fred

You can input a value or a string from the command line with the 'input()' function. This has two forms depending on whether you want to input a number or a string:

```
yval=input('Enter a number: ');
name=input('Enter your name: ', 's');
```

Input and Output Commands

MATLAB provides the following input and output related commands −

Command	Purpose
disp	Displays contents of an array or string.
fscanf	Read formatted data from a file.
format	Controls screen-display format.
fprintf	Performs formatted writes to screen or file.
input	Displays prompts and waits for input.
;	Suppresses screen printing.

The **fscanf** and **fprintf** commands behave like C scanf and printf functions. They support the following format codes

Register Number_____

[16CSU304B]

KARPAGAM ACADEMY FOR HIGHER EDUCATION
KARPAGAM UNIVERSITY
(Deemed University Established Under Section 3 of UGC Act 1956)
Coimbatore - 641021.
(For the candidates admitted from 2016 onwards)

THIRD INTERNAL EXAMINATION, SEPTEMBER 2017
Third Semester
COMPUTER SCIENCE
PROGRAMMING IN MATLAB

Date & Session : 14.09.2017 **Class: II B.Sc CS**
Maximum : 50 Marks **Duration: 2 Hours**

PART-A (20 X 1 = 20 Marks)
(Answer ALL the Questions)

1. The _____ construct is a special form branching construct designed to trap errors
a) try/catch b)switch c)if d)if else

2. The statements in the _____ block will only be executed of an error occurred in try block
a)catch b)else c)try d)if else

3. The _____ loop is a loop that executes a block of statements a specified number of times
a)do **b)while** c)whiledo d) for

4. The _____ statement terminates the current pass through the loop and return control to the top of the loop
a)break **b)continue** c)skip d)end

5. _____ is used to terminate the program due to incorrect input and gives the error message
a)break b)continue **c)error** d)none

6. The _____ statement terminates execution of **for** or **while** loop
a)for loop **b)break** c)continue d)nested loop

7. The _____ loop repeatedly executes statements while condition is true.
a) for loop b)for **c)while** d)nested loop

8. _____ is a repetition statement
a)for b)else if c)switch d)nested if

9. The while-end loop is classified as a/an
a)definite loop **b)indefinite loop**. c) infinite loop. d)ridiculous loop.

10. If _____ loops are nested, they should have independent loop index variables.
a)do while b)while c)if **d)for**

11. A program that response to event is said to be _____
a)program driven **b)event driven** c)events d)none

12. A most common container is a _____

a)figure b)workspace c)plot d)area

13. _____ may be modal or non modal

a)toolbox **b)dialog boxes** c)toggle button d)menus

14. Each graphics object is known by a unique number called a _____

a)Handle b)object c)term d)component

15. GUI Stands for _____

a)Graphical User Input **b)Graphical User Interface**

c)Geometric User Interface d)Graphical Unique Interface

16. _____ can contain components or other containers

a)callbacks **b)panel** c)button group d)component

17. A _____ allows a user to select actions without additional components appearing on the GUI display

a)tools b)list box **c)menus** d)dialog boxes

18. Each figure can contain _____ types of objects

a)six b)eight c)two **d)seven**

19. The _____ function is the inverse of fprintf().

a)fscanf() b)fgetl() c)object d)Menu Editor

20. We can add menu options to your dialog with the _____

a)Figures **b)Menu Editor** c)Edit boxes d)GUI

PART-B (3 X 2 = 6 Marks)
(Answer ALL the Questions)

21. What is If Statement and its syntax?

Execute statements if condition is true

Syntax

```
if expression
   statements
elseif expression
   statements
else
   statements
end
```

Description

if *expression*, *statements*, end evaluates an expression, and executes a group of statements when the expression is true. An expression is true when its result is nonempty and contains only nonzero elements (logical or real numeric). Otherwise, the expression is false.

22. What is Manipulating a text?

Manipulating text includes the following:

1. Writing to a text file
2. Reading from a text file
3. Randomising and sorting a list
4. Searching a list

23. What are the types of loops does Matlab provides?

For loop
While loop
Nested loop

PART-C (3 X 8 = 24 Marks)
(Answer ALL the Questions)

24. a) Explain Conditional statements with example.

CONDITIONAL STATEMENTS

MATLAB provides following types of decision making statements. Click the following links to check their detail −

Statement	Description
if ... end statement	An **if ... end statement** consists of a boolean expression followed by one or more statements.
if...else...end statement	An **if statement** can be followed by an optional **else statement**, which executes when the boolean expression is false.
If... elseif...elseif...else...end statements	An **if** statement can be followed by one (or more) optional **elseif...** and an **else** statement, which is very useful to test various conditions.
nested if statements	You can use one **if** or **elseif** statement inside another **if** or **elseif** statement(s).
switch statement	A **switch** statement allows a variable to be tested for equality against a list of values.
nested switch statements	You can use one **switch** statement inside another **switch** statement(s).

If end

An **if ... end** statement consists of an **if** statement and a boolean expression followed by one or more statements. It is delimited by the **end** statement.

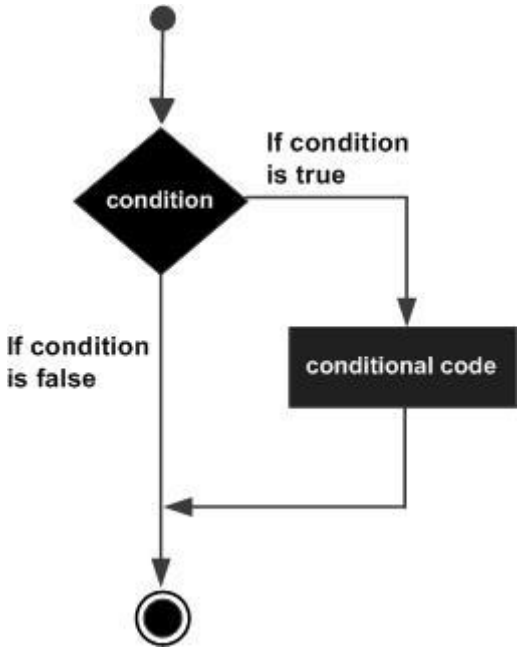
Syntax

The syntax of an if statement in MATLAB is −

```
if <expression>
% statement(s) will execute if the boolean expression is true
<statements>
end
```

If the expression evaluates to true, then the block of code inside the if statement will be executed. If the expression evaluates to false, then the first set of code after the end statement will be executed.

Flow Diagram



Example

Create a script file and type the following code −

```
a = 10;
% check the condition using if statement
  if a < 20
  % if condition is true then print the following
    fprintf('a is less than 20\n' );
  end
fprintf('value of a is : %d\n', a);
```

When you run the file, it displays the following result −

```
a is less than 20
value of a is : 10
```

If else end

An if statement can be followed by an optional else statement, which executes when the expression is false.

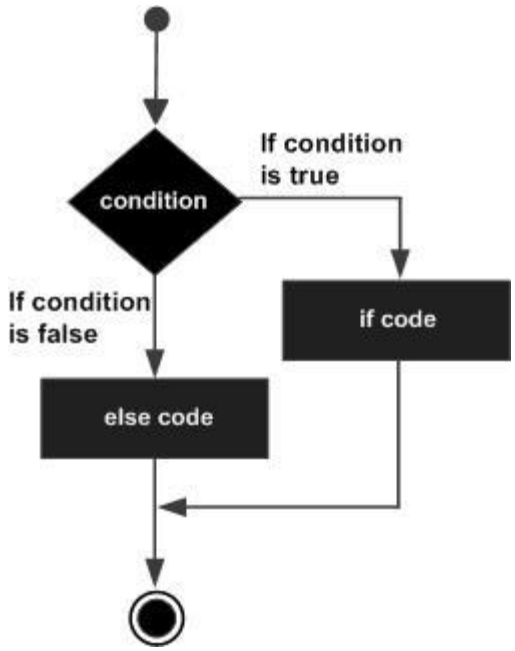
Syntax

The syntax of an if...else statement in MATLAB is −

```
if <expression>
% statement(s) will execute if the boolean expression is true
<statement(s)>
else
<statement(s)>
% statement(s) will execute if the boolean expression is false
end
```

If the boolean expression evaluates to true, then the if block of code will be executed, otherwise else block of code will be executed.

Flow Diagram



Example

Create a script file and type the following code −

```
a = 100;
% check the boolean condition
  if a < 20
    % if condition is true then print the following
    fprintf('a is less than 20\n' );
  else
    % if condition is false then print the following
    fprintf('a is not less than 20\n' );
  end
  fprintf('value of a is : %d\n', a);
```

When the above code is compiled and executed, it produces the following result −

```
a is not less than 20
value of a is : 100
```

If elseif elseif else end statements

An **if** statement can be followed by one (or more) optional **elseif...** and an **else** statement, which is very useful to test various conditions.

When using if... elseif...else statements, there are few points to keep in mind:

- An if can have zero or one else's and it must come after any elseif's.
- An if can have zero to many elseif's and they must come before the else.
- Once an else if succeeds, none of the remaining elseif's or else's will be tested.

Syntax

```
if <expression 1>
% Executes when the expression 1 is true
<statement(s)>

elseif <expression 2>
% Executes when the boolean expression 2 is true
<statement(s)>

Elseif <expression 3>
% Executes when the boolean expression 3 is true
<statement(s)>

else
%  executes when the none of the above condition is true
<statement(s)>
end
```

Example

Create a script file and type the following code in it −

```
a = 100;
```

```
%check the boolean condition
  if a == 10
    % if condition is true then print the following
    fprintf('Value of a is 10\n' );
  elseif( a == 20 )
    % if else if condition is true
    fprintf('Value of a is 20\n' );
  elseif a == 30
    % if else if condition is true
    fprintf('Value of a is 30\n' );
  else
    % if none of the conditions is true '
    fprintf('None of the values are matching\n');
  fprintf('Exact value of a is: %d\n', a );
  end
```

When the above code is compiled and executed, it produces the following result −

```
None of the values are matching
Exact value of a is: 100
```

Nested If Statements

It is always legal in MATLAB to nest if-else statements which means you can use one if or elseif statement inside another if or elseif statement(s).

Syntax

The syntax for a nested if statement is as follows −

```
if <expression 1>
% Executes when the boolean expression 1 is true
  if <expression 2>
    % Executes when the boolean expression 2 is true
 end
```

```
end
```

You can nest elseif...else in the similar way as you have nested if statement.

Example

Create a script file and type the following code in it −

```
a = 100;
b = 200;
  % check the boolean condition
  if( a == 100 )

    % if condition is true then check the following
    if( b == 200 )

      % if condition is true then print the following
      fprintf('Value of a is 100 and b is 200\n' );
    end

  end
  fprintf('Exact value of a is : %d\n', a );
  fprintf('Exact value of b is : %d\n', b );
```

When you run the file, it displays −

```
Value of a is 100 and b is 200
Exact value of a is : 100
Exact value of b is : 200
```

Switch Statements

A switch block conditionally executes one set of statements from several choices. Each choice is covered by a case statement.

An evaluated switch_expression is a scalar or string.

An evaluated case_expression is a scalar, a string or a cell array of scalars or strings.

The switch block tests each case until one of the cases is true. A case is true when −

- For numbers, **eq(case_expression,switch_expression)**.
- For strings, **strcmp(case_expression,switch_expression)**.
- For objects that support the **eq(case_expression,switch_expression)**.
- For a cell array case_expression, at least one of the elements of the cell array matches switch_expression, as defined above for numbers, strings and objects.

When a case is true, MATLAB executes the corresponding statements and then exits the switch block.

The **otherwise** block is optional and executes only when no case is true.

Syntax

The syntax of switch statement in MATLAB is −

```
switch <switch_expression>
  case <case_expression>
    <statements>
  case <case_expression>
    <statements>
    ...
    ...
  otherwise
    <statements>
end
```

Example

Create a script file and type the following code in it −

```
grade = 'B';
  switch(grade)
  case 'A'
     fprintf('Excellent!\n' );
  case 'B'
     fprintf('Well done\n' );
  case 'C'
     fprintf('Well done\n' );
  case 'D'
     fprintf('You passed\n' );

  case 'F'
     fprintf('Better try again\n' );

  otherwise
     fprintf('Invalid grade\n' );
  end
```

When you run the file, it displays −

```
Well done
```

Nested Switch statements

It is possible to have a switch as part of the statement sequence of an outer switch. Even if the case constants of the inner and outer switch contain common values, no conflicts will arise.

Syntax

The syntax for a nested switch statement is as follows −

```
switch(ch1)
  case 'A'
```

```
   fprintf('This A is part of outer switch');
      switch(ch2)
         case 'A'
         fprintf('This A is part of inner switch' );

         case 'B'
         fprintf('This B is part of inner switch' );
      end
   case 'B'
   fprintf('This B is part of outer switch' );
end
```

Example

Create a script file and type the following code in it −

```
a = 100;
b = 200;
switch(a)
   case 100
   fprintf('This is part of outer switch %d\n', a );

   switch(b)
      case 200
      fprintf('This is part of inner switch %d\n', a );
   end
end
fprintf('Exact value of a is : %d\n', a );
fprintf('Exact value of b is : %d\n', b );
```

When you run the file, it displays −

```
This is part of outer switch 100
```

```
This is part of inner switch 100
Exact value of a is : 100
Exact value of b is : 200
```

[OR]

b) Explain in detail while, for loop with example.

Loop Control Statements

Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.

MATLAB supports the following control statements. Click the following links to check their detail.

Control Statement	Description
break statement	Terminates the **loop** statement and transfers execution to the statement immediately following the loop.
continue statement	Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.

While Loop

The while loop repeatedly executes statements while condition is true.

Syntax

The syntax of a while loop in MATLAB is −

```
while <expression>
  <statements>
end
```

The while loop repeatedly executes program statement(s) as long as the expression remains true.

An expression is true when the result is nonempty and contains all nonzero elements (logical or real numeric). Otherwise, the expression is false.

Example

Create a script file and type the following code −

```
a = 10;
% while loop execution
while( a < 20 )
  fprintf('value of a: %d\n', a);
  a = a + 1;
end
```

When you run the file, it displays the following result −

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

For Loop

A **for loop** is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.

Syntax

The syntax of a **for loop** in MATLAB is −

```
for index = values
   <program statements>
          ...
end
```

values has one of the following forms −

Format	Description
initval:endval	increments the index variable from *initval* to *endval* by 1, and repeats execution of *program statements* until *index* is greater than *endval*.
initval:step:endval	increments *index* by the value step on each iteration, or decrements when step is negative.
valArray	creates a column vector *index* from subsequent columns of array *valArray* on each iteration. For example, on the first iteration, index = valArray(:,1). The loop executes for a maximum of n times, where n is the number of columns of *valArray*, given by numel(valArray, 1, :). The input *valArray*can be of any MATLAB data type, including a string, cell array, or struct.

Example 1

Create a script file and type the following code −

```
for a = 10:20
   fprintf('value of a: %d\n', a);
end
```

When you run the file, it displays the following result −

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
value of a: 20
```

Example 2

Create a script file and type the following code −

```
for a = [24,18,17,23,28]
  disp(a)
end
```

When you run the file, it displays the following result −

```
24

18

17

23

28
```

Nested Loop

MATLAB allows to use one loop inside another loop. Following section shows few examples to illustrate the concept.

Syntax

The syntax for a nested for loop statement in MATLAB is as follows −

```
for m = 1:j
  for n = 1:k
    <statements>;
  end
end
```

The syntax for a nested while loop statement in MATLAB is as follows −

```
while <expression1>
  while <expression2>
```

```
    <statements>
  end
end
```

Example

Let us use a nested for loop to display all the prime numbers from 1 to 100. Create a script file and type the following code −

```
for i=2:100
  for j=2:100
    if(~mod(i,j))
      break; % if factor found, not prime
    end
  end
  if(j > (i/j))
    fprintf('%d is prime\n', i);
  end
end
```

When you run the file, it displays the following result −

```
2 is prime
3 is prime
5 is prime
7 is prime
11 is prime
13 is prime
17 is prime
19 is prime
23 is prime
29 is prime
31 is prime
37 is prime
41 is prime
43 is prime
47 is prime
53 is prime
```

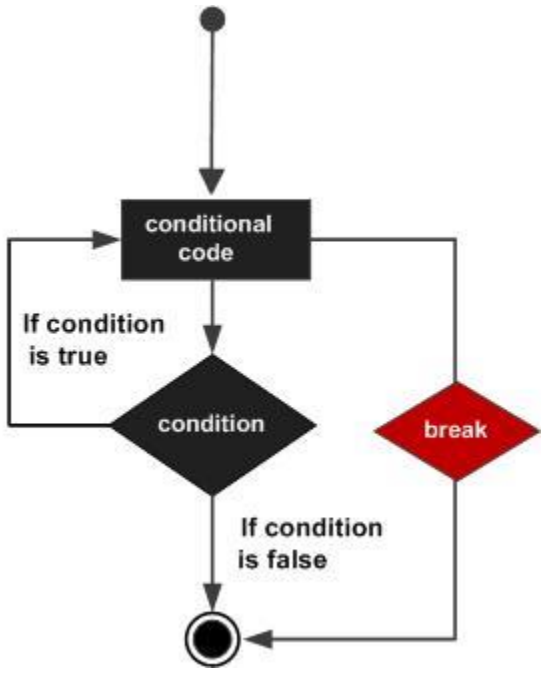
```
59 is prime
61 is prime
67 is prime
71 is prime
73 is prime
79 is prime
83 is prime
89 is prime
97 is prime
```

Break Statement

The break statement terminates execution of **for** or **while** loop. Statements in the loop that appear after the break statement are not executed.

In nested loops, break exits only from the loop in which it occurs. Control passes to the statement following the end of that loop.

Flow Diagram



Example

Create a script file and type the following code:

```
a = 10;
% while loop execution
while (a < 20 )
  fprintf('value of a: %d\n', a);
  a = a+1;
    if( a > 15)
      % terminate the loop using break statement
      break;
    end
end
```

When you run the file, it displays the following result:

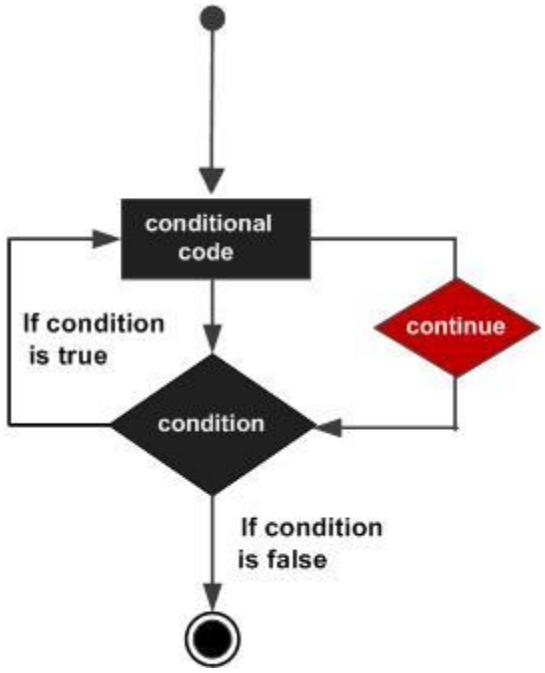
```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
```

Continue Statements

The continue statement is used for passing control to next iteration of for or while loop.

The continue statement in MATLAB works somewhat like the break statement. Instead of forcing termination, however, 'continue' forces the next iteration of the loop to take place, skipping any code in between.

Flow Diagram



Example

Create a script file and type the following code −

```
a = 10;
%while loop execution
while a < 20
  if a == 15
    % skip the iteration
    a = a + 1;
    continue;
  end
fprintf('value of a: %d\n', a);
a = a + 1;
end
```

When you run the file, it displays the following result −

```
value of a: 10
value of a: 11
value of a: 12
```

```
value of a: 13
value of a: 14
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

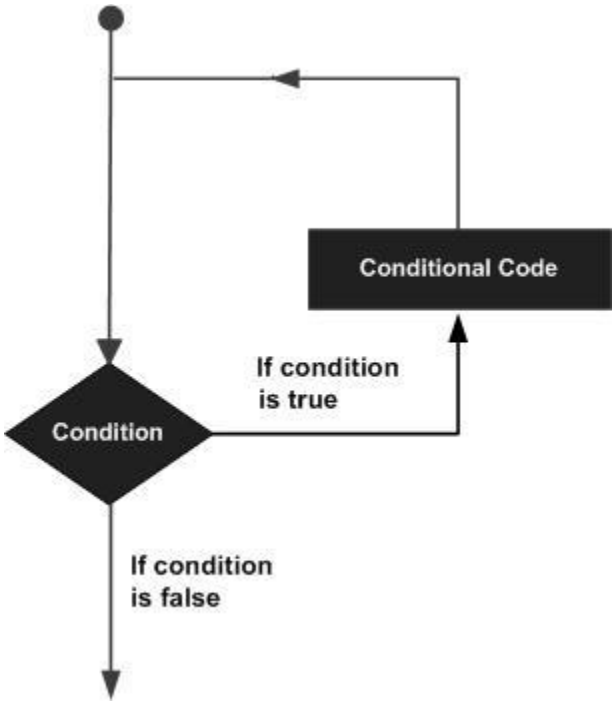
25. a)Briefly describe about Repetition Statement with example.

REPETITION STATEMENTS

There may be a situation when you need to execute a block of code several number of times. In general, statements are executed sequentially. The first statement in a function is executed first, followed by the second, and so on.

Programming languages provide various control structures that allow for more complicated execution paths.

A loop statement allows us to execute a statement or group of statements multiple times and following is the general form of a loop statement in most of the programming languages −



MATLAB provides following types of loops to handle looping requirements. Click the following links to check their detail −

Loop Type	Description
while loop	Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.
for loop	Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.
nested loops	You can use one or more loops inside any another loop.

Loop Control Statements

Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.

MATLAB supports the following control statements. Click the following links to check their detail.

Control Statement	Description
break statement	Terminates the **loop** statement and transfers execution to the statement immediately following the loop.
continue statement	Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.

While Loop

The while loop repeatedly executes statements while condition is true.

Syntax

The syntax of a while loop in MATLAB is −

```
while <expression>
  <statements>
end
```

The while loop repeatedly executes program statement(s) as long as the expression remains true.

An expression is true when the result is nonempty and contains all nonzero elements (logical or real numeric). Otherwise, the expression is false.

Example

Create a script file and type the following code −

```
a = 10;
% while loop execution
while( a < 20 )
  fprintf('value of a: %d\n', a);
  a = a + 1;
end
```

When you run the file, it displays the following result −

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

For Loop

A **for loop** is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.

Syntax

The syntax of a **for loop** in MATLAB is −

```
for index = values
```

```
   <program statements>
         ...
end
```

values has one of the following forms −

Format	Description
initval:endval	increments the index variable from *initval* to *endval* by 1, and repeats execution of *program statements* until *index* is greater than *endval*.
initval:step:endval	increments *index* by the value step on each iteration, or decrements when step is negative.
valArray	creates a column vector *index* from subsequent columns of array *valArray* on each iteration. For example, on the first iteration, index = valArray(:,1). The loop executes for a maximum of n times, where n is the number of columns of *valArray*, given by numel(valArray, 1, :). The input *valArray*can be of any MATLAB data type, including a string, cell array, or struct.

Example 1

Create a script file and type the following code −

```
for a = 10:20
   fprintf('value of a: %d\n', a);
end
```

When you run the file, it displays the following result −

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
```

```
value of a: 16
value of a: 17
value of a: 18
value of a: 19
value of a: 20
```

Example 2

Create a script file and type the following code −

```
for a = [24,18,17,23,28]
   disp(a)
end
```

When you run the file, it displays the following result −

```
24

18

17

23

28
```

Nested Loop

MATLAB allows to use one loop inside another loop. Following section shows few examples to illustrate the concept.

Syntax

The syntax for a nested for loop statement in MATLAB is as follows −

```
for m = 1:j
   for n = 1:k
      <statements>;
   end
```

```
end
```

The syntax for a nested while loop statement in MATLAB is as follows −

```
while <expression1>
  while <expression2>
    <statements>
  end
end
```

Example

Let us use a nested for loop to display all the prime numbers from 1 to 100. Create a script file and type the following code −

```
for i=2:100
  for j=2:100
    if(~mod(i,j))
      break; % if factor found, not prime
    end
  end
  if(j > (i/j))
    fprintf('%d is prime\n', i);
  end
end
```

When you run the file, it displays the following result −

```
2 is prime
3 is prime
5 is prime
7 is prime
11 is prime
13 is prime
17 is prime
19 is prime
23 is prime
```

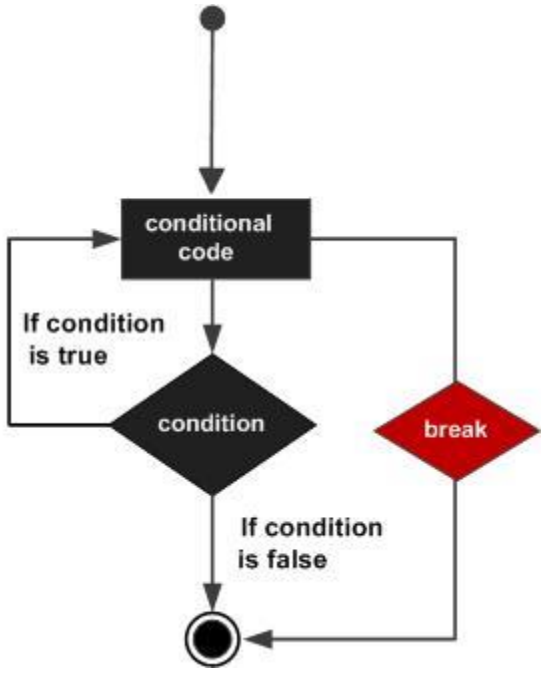
```
29 is prime
31 is prime
37 is prime
41 is prime
43 is prime
47 is prime
53 is prime
59 is prime
61 is prime
67 is prime
71 is prime
73 is prime
79 is prime
83 is prime
89 is prime
97 is prime
```

Break Statement

The break statement terminates execution of **for** or **while** loop. Statements in the loop that appear after the break statement are not executed.

In nested loops, break exits only from the loop in which it occurs. Control passes to the statement following the end of that loop.

Flow Diagram



Example

Create a script file and type the following code:

```
a = 10;
% while loop execution
while (a < 20 )
  fprintf('value of a: %d\n', a);
  a = a+1;
    if( a > 15)
      % terminate the loop using break statement
      break;
    end
end
```

When you run the file, it displays the following result:

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
```

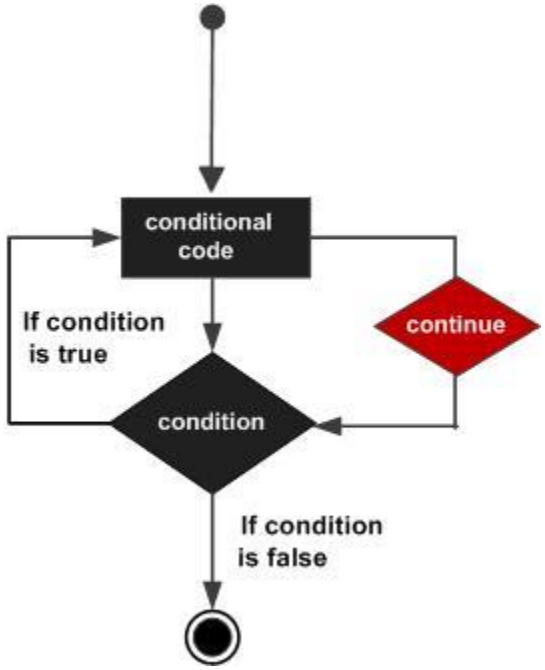
```
value of a: 14
value of a: 15
```

Continue Statements

The continue statement is used for passing control to next iteration of for or while loop.

The continue statement in MATLAB works somewhat like the break statement. Instead of forcing termination, however, 'continue' forces the next iteration of the loop to take place, skipping any code in between.

Flow Diagram



Example

Create a script file and type the following code −

```
a = 10;
%while loop execution
while a < 20
  if a == 15
    % skip the iteration
    a = a + 1;
```

```
    continue;
  end
fprintf('value of a: %d\n', a);
a = a + 1;
end
```

When you run the file, it displays the following result −

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

[OR]

b)Discuss about Manipulating a text in detail with example

MANIPULATING TEXT

1. Writing to a text file

To save the results of some computation to a file in text format reqires the following steps:

a. Open a new file, or overwrite an old file, keeping a 'handle' for the file.
b. Print the values of expressions to the file, using the file handle
c. Close the file, using the file handle

The file handle is a just a variable which identifies the open file in your program. This allows you to have any number of files open at any one time.

```
% open file
fid = fopen('myfile.txt','wt');     % 'wt' means "write text"
if (fid < 0)
   error('could not open file "myfile.txt"');
end;
% write some stuff to file
for i=1:100
   fprintf(fid,'Number = %3d Square = %6d\n',i,i*i);
end;
% close the file
fclose(fid);
```

2. Reading from a text file

To read some results from a text file is straightforward if you just want to load the whole file into memory. This requires the following steps:

a. Open an existing file, keeping a 'handle' for the file.
b. Read expressions from the file into a single array, using the file handle
c. Close the file, using the file handle

The fscanf() function is the inverse of fprintf(). However it returns the values it reads as values in a matrix. You can control the 'shape' of the output matrix with a third argument.

```
A = fscanf(fid,"%g %g %g\n",[3,inf])      % A has 3 rows and 1 col per line
disp(A(1,1))        % display first value on first line
disp(A(1,2))        % display first value on second line
disp(A(2,1))        % display second value on first line
```

Thus to read back the data we saved above:

```
% open file
fid = fopen('myfile.txt','rt');     % 'rt' means "read text"
if (fid < 0)
   error('could not open file "myfile.txt"');
end;
% read from file into table with 2 rows and 1 column per line
tab = fscanf(fid,'Number = %d Square = %d\n',[2,inf]);
% close the file
fclose(fid);
rtab = tab';         % convert to 2 columns and 1 row per line
```

Reading a table of strings is more complex, since the strings have to be the same length. We can use the fgetl() function to get a line of text as characters, but we'll first need to find out the length of the longest string, then ensure all strings are the same length. Here is a complete function for loading a text file as a table of fixed-length strings:

```
function tab=readtextfile(filename)
% Read a text file into a matrix with one row per input line
% and with a fixed number of columns, set by the longest line.
% Each string is padded with NUL (ASCII 0) characters
%
% open the file for reading
ip = fopen(filename,'rt');         % 'rt' means read text
if (ip < 0)
   error('could not open file');   % just abort if error
end;
% find length of longest line
max=0;                          % record length of longest string
cnt=0;                         % record number of strings
```

```
s = fgetl(ip);                % get a line
while (ischar(s))             % while not end of file
  cnt = cnt+1;
  if (length(s) > max)        % keep record of longest
      max = length(s);
  end;
   s = fgetl(ip);             % get next line
end;
% rewind the file to the beginning
frewind(ip);
% create an empty matrix of appropriate size
tab=char(zeros(cnt,max));        % fill with ASCII zeros
% load the strings for real
cnt=0;
s = fgetl(ip);
while (ischar(s))
  cnt = cnt+1;
  tab(cnt,1:length(s)) = s;      % slot into table
   s = fgetl(ip);
end;
% close the file and return
fclose(ip);
return;
```

Here is an example of its use:

```
% write some variable length strings to a file
op = fopen('weekdays.txt','wt');
fprintf(op,'Sunday\nMonday\nTuesday\nWednesday\n');
fprintf(op,'Thursday\nFriday\nSaturday\n');
fclose(op);
% read it into memory
tab = readtextfile('weekdays.txt');
% display it
disp(tab);
```

3. Randomising and sorting a list

Assuming we have a table of values, how can we randomise the order of the entries? A good way of achieving this is analogous to shuffling a pack of cards. We pick two positions in the pack, then swap over the cards at those two positions. We then just repeat this process enough times that each card is likely to be swapped at least once.

```
function rtab=randomise(tab)
% randomise the order of the rows in tab.
% columns are unaffected
[nrows,ncols]=size(tab);        % get size of input matrix
```

```
cnt = 10*nrows;                 % enough times
while (cnt > 0)
   pos1 = 1+fix(nrows*rand);     % get first random row
   pos2 = 1+fix(nrows*rand);     % get second random row
   tmp = tab(pos1,:);            % save first row
   tab(pos1,:) = tab(pos2,:);    % swap second into first
   tab(pos2,:) = tmp;            % move first into second
   cnt=cnt-1;
end;
rtab=tab;                       % return randomised table
return;
```

Sorting a list is easy if you just want some standard alphabetic ordering. But what if you want to choose some arbitrary ordering function? For example, how could you sort strings when case was not important? Here we use the ability of MATLAB to evaluate a function by name (feval()) so that we can provide the name of a function for doing the comparisons the way we want. This function should take two rows and return –1 if the first row sorts earlier than the second, 1 if the second row sorts earlier than the first and 0 if there is no preference. Here is a case-independent comparison function:

```
function flag=comparenocase(str1,str2)
% compares two strings without regard to case
% returns –1, 0, 1 if str1 is less than, equal, greater than str2.
len1=length(str1);
len2=length(str2);
for i=1:min(len1,len2)
   c1 = str1(i);
   c2 = str2(i);
   if (('a' <= c1)&(c1 <= 'z'))
      c1 = char(abs(c1)-32);        % convert lower case to upper
   end;
   if (('a' <= c2)&(c2 <= 'z'))
      c2 = char(abs(c2)-32);        % convert lower case to upper
   end;
   if (c1 < c2)
      flag = -1;                    % str1 sorts earlier
      return;
   elseif (c2 < c1)
      flag = 1;                     % str2 sorts earlier
      return;
   end;
end;
% strings match up to length of shorter, so
if (len1 < len2)
   flag = -1;                       % str1 sorts earlier
```

```
elseif (len2 < len1)
   flag = 1;                    % str2 sorts earlier
else
   flag = 0;                    % no preference
end;
return;
```

Here is a sort function that might be used with this comparison function.

```
function stab=functionsortrows(tab,funcname)
% sorts the rows of the input table using the supplied
% function name to provide an ordering on pairs of rows
[nrows,ncols]=size(tab);
for i=2:nrows                    % sort each row into place
   j = i;
   tmp = tab(j,:);               % save row
   % compare this row with higher rows to see where it goes
   while ((j > 1)&(feval(funcname,tmp,tab(j-1,:))<0))
      tab(j,:) = tab(j-1,:);     % shift higher rows down
      j = j - 1;
   end;
   tab(j,:) = tmp;               % put in ordered place
end;
stab = tab;                      % return sorted table
return;
```

4. Searching a list

How might we search a list of items for an item matching a specific value? If the list is unordered, all we can do is run down the list testing each entry in turn. This function finds the index of a row in a table that contains (anywhere) the characters in the supplied match string:

```
function idx=findstring(tab,str)
% find the row index containing a matching string
% returns 0 if the string is not found
[nrows,ncols]=size(tab);
for idx=1:nrows
   matches = findstr(tab(idx,:),str);
   if (length(matches)>0)
      return;
   end;
end;
idx=0;
return;
```

However, the process can be much faster if the listed is sorted and we are searching for an exact match only. A so-called binary search is the fastest possible way of finding an item in a sorted list:

```
function idx=binarysearch(tab,val)
% returns the row index of val in sorted table tab
% returns 0 if val is not found
[nrows,ncols]=size(tab);
lo=1;
hi=nrows;
while (lo <= hi)
   idx = fix(lo+hi)/2;
   if (val < tab(idx,:))
      hi = idx - 1;
   elseif (val > tab(idx,:))
      lo = idx + 1;
   else
      return;
   end;
end;
idx=0;
return;
```

26. a)Explain about GUI Interface in detail.

GUI INTERFACE

1. **Elements of a Graphical User Interface**

By a graphical user interface, we mean that we can give a MATLAB program the look and feel of a typical Windows application. The MATLAB GUI design system allows you to create applications consisting of one or more 'dialogs' containing typical 'controls' such as buttons, edit boxes, lists and pictures.

One of the important aspects of a Windows application that is unlike the kind of programs we have considered up to now is that they interact asynchronously with the user. The user can select any function of the program at any time. This means that you need to store the 'state' of your program in a set of variables and be prepared to execute any function based on the current state at any time.

The MATLAB GUI design system helps you in this by associating functions with each element of the dialog. Thus when you press a button, click on a menu, or enter a number in an edit box, you can arrange for a function in your program to be called. Your task is to program the actions related to that function, e.g. opening a file, playing a sound, or displaying the results of a calculation.

The most common controls are:

- Menu options. Selection calls up an operation by name.
- Push buttons. Clicking calls up some operation.
- Edit boxes. User can enter some text or numerical value.
- List boxes. User can choose among list of items.

- Figures. Program can display graphical results.
- Text. Program can display textual result.

You can use the controls themselves to store data or you can create a set of global variables.

2. How to build a simple dialogue

To start the design program type 'guide' at the MATLAB prompt. You are presented with a blank form upon which you can position controls. Choose a control from the palette and click and size the control on the page to position it. Each control is automatically given a name based on its type.

When the layout is complete, you can save the design to a '.fig' file. This will automatically create a matching '.m' program file which you can use to launch the application and store the code that is operated by the controls. It is not necessary to store all your code in the matching '.m' file; indeed it is a good idea to break up any large sections of code into its own function blocks stored in separate files. You will see that the layout designer builds a 'callback' function prototype in the program file for each control that provides input to the application. This function will be called automatically when that control is activated.

You can edit the properties of the controls on the layout editor by right-clicking on them and choosing 'Property Inspector'. In particular the 'String' property is used to store the default text for buttons, list boxes and edit boxes. The 'Tag' property is the name of the control; and until you are familiar with MATLAB, it is advisable not to change the default name. You can also use the Property Inspector to change the name of the dialog itself.

You can add menu options to your dialog with the 'Menu Editor'. If you leave the callback function entry as "%automatic", then the menu editor adds callback functions to your program for each menu item. Otherwise create your own callback function using existing ones as a model, and associate a call to the function with the menu item manually.

It is important to realise that the '.m' file associated with your application is executed afresh each time there is some event in the dialog. That is you must store the 'current state' of the program in global variables in the workspace, and not in variables local to a function. You can ensure this by using a 'global'statement and initialising them in the part of the file where the figure is initialised.

You can access any property of any control using the 'Tag' property of the control and the MATLAB 'get()' and 'set()' functions.

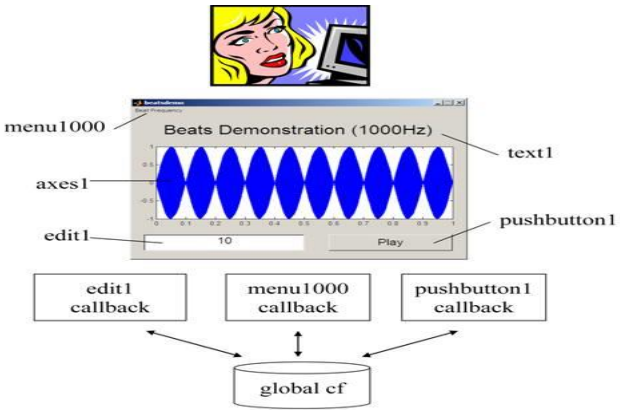
```
value = get(handles.ControlTagName,'PropertyName');
set(handles.ControlTagName,'PropertyName','Value');
```

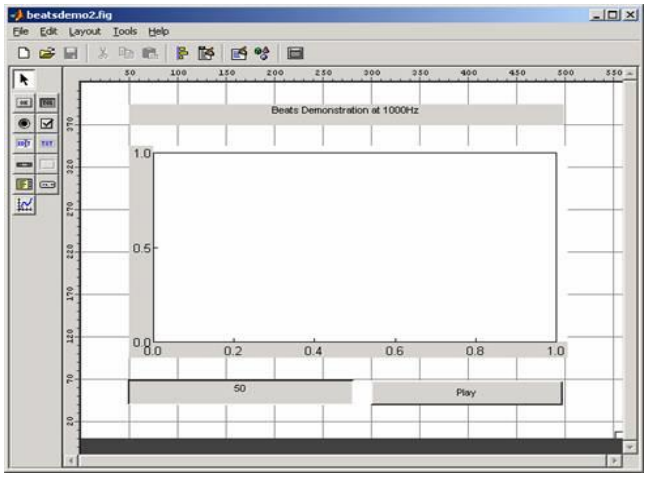
For example:

```
text = get(handles.edit1,'String');
set(handles.edit1,'String','100');
```

Note that most properties have to be get() and set() as strings. Use the num2str() and str2num() functions to help convert between strings and numeric values.

3. Worked example





[OR]

b)Explain about attaching buttons to actions, Getting Input and Output.

GETTING INPUT, GETTING OUTPUT

uicontrol

Create user interface control object

Syntax

```
c = uicontrol
c = uicontrol(Name,Value,...)
c = uicontrol(parent)
c = uicontrol(parent,Name,Value,...)
uicontrol(c)
```

Description

`c = uicontrol` creates a uicontrol (push button) in the current figure and returns the uicontrol object, `c`. If there is no figure available, then MATLAB® creates a new figure to serve as the parent.

`c = uicontrol(Name,Value,...)` creates a uicontrol and specifies one or more uicontrol property names and corresponding values. Use this syntax to override the default uicontrol properties. The default uicontrol style is `'pushbutton'`.

`c = uicontrol(parent)` creates a uicontrol and designates a specific parent object. The `parent` argument can be a figure, uipanel, uibuttongroup, or uitab object.

`c = uicontrol(parent,Name,Value,...)` creates a uicontrol with a specific parent and one or more uicontrol properties.

`uicontrol(c)` gives focus to a specific uicontrol object, `c`.

Specifying the Uicontrol Style

- When selected, most uicontrol objects perform a predefined action. To create a specific type of uicontrol, set the `Style`property as one of the following values. You can specify part of the `Style` value if it is unique among all the styles. For example, instead of `'radiobutton'`, you can specify `'radio'`.
- `'checkbox'` – A check box generates an action when you select it. Use check boxes to provide a number of independent choices. To activate a check box, click the mouse button on the object. The check box updates its appearance when its state changes.
- `'edit'` – Editable text fields enable you to enter or modify text values. Use editable text when you want free text as input. To enable multiple lines of text, set `Max-Min>1`. Multiline edit boxes provide a vertical scroll bar for scrolling. The arrow keys also provide a way to scroll. Obtain the current text by getting the `String` property. The `String` property does not update as you type in an edit box. To execute the callback routine for an edit text control, type in the desired text and then do one of the following:
    - Click another component, the menu bar, or elsewhere on the window.
    - For a single line editable text box, press **Enter**.
    - For a multiline editable text box, press **Ctl+Enter**.
- `'frame'`
- `'listbox'` – List boxes display a list of items, from which you can select one or more items. Unlike pop-up menus, list boxes do not expand when clicked. The `Min` and `Max` properties control the selection mode:
    - To enable multiple selection of items, set `Max-Min > 1`.
    - To enable selection of only one item at a time, set `Max-Min <= 1`

- The `Value` property stores the row indexes of currently selected list box items, and is a vector value when you select multiple items. After any mouse button up event that changes the `Value` property, MATLAB evaluates the list box's callback routine. To delay action when multiple items can be selected, you can associate a "Done" push button with the list box. Use the callback for that button to evaluate the list box `Value` property.
- List boxes with the `Enable` property set to `on` differentiate between single and double left clicks. MATLAB sets the figure `SelectionType` property to `normal` or `open` accordingly before evaluating the list box `Callback` property. For enabled list boxes, **Ctrl**-left click and **Shift**-left click also set the figure `SelectionType` property to `normal` or `open`, respectively indicating a single or double click.
- `'popupmenu'` – Pop-up menus (also known as drop-down menus) display a list of choices when you open them with a button-press. When closed, a pop-up menu indicates the current choice. Pop-up menus are useful when you want to provide a number of mutually exclusive choices, but do not want to take up the amount of space that a group of radio buttons requires.
- `'pushbutton'` – Push buttons generate an action when activated. Left-click a push button to activate it. The button appears to depress until you release the mouse button. The callback activates when you release the mouse button while still pointing within the push button.
- `'radiobutton'` – Radio buttons are similar to check boxes, but are intended to be mutually exclusive within a group of related radio buttons. When used this way, you can only select one radio button at any given time. To activate a radio button, click and release the mouse button over it. The easiest way to implement mutually exclusive behavior for a set of radio buttons is to place them within a `uibuttongroup`.
- `'slider'` – Sliders accept numeric input within a specific range when you move the “thumb” button along a bar. The location of the thumb indicates a numeric value, assigned to the `Value` property when you release the mouse button. You can set the minimum, maximum, and current values, and step sizes of a slider.
- Move the thumb by doing any one of the following:
    - Press the mouse button on the thumb, and drag it along the bar.
    - Click in the bar or on arrow buttons located at both ends of the bar.
    - Click the keyboard arrow keys when the slider is in focus.
- `'text'` – Static text boxes display lines of text. You typically use static text to label other controls, provide information to the user, or indicate values associated with a slider. If you assign the `Callback` property of a static text object to a function (or a character vector containing a MATLAB command), the static text will not respond when users try to interact with the text. However, you can code the `ButtonDownFcn` callback to respond to mouse clicks on the static text. See Tips for more information.

- `'togglebutton'` – Toggle buttons are similar in appearance to push buttons, but they visually indicate their state, either `'on'` (depressed) or `'off'` (up). Clicking a toggle button changes its state, and switches its `Value` property between the toggle button's `Min` and `Max` values.
- **Examples**

- Create uicontrols to allow users to adjust the appearance of a plot. For instance, create a program file called `myui.m` that contains the following code.

```
function myui
  % Create a figure and axes
  f = figure('Visible','off');
  ax = axes('Units','pixels');
  surf(peaks)

  % Create pop-up menu
  popup = uicontrol('Style', 'popup',...
        'String', {'parula','jet','hsv','hot','cool','gray'},...
        'Position', [20 340 100 50],...
        'Callback', @setmap);

 % Create push button
  btn = uicontrol('Style', 'pushbutton', 'String', 'Clear',...
     'Position', [20 20 50 20],...
     'Callback', 'cla');

 % Create slider
  sld = uicontrol('Style', 'slider',...
     'Min',1,'Max',50,'Value',41,...
     'Position', [400 20 120 20],...
     'Callback', @surfzlim);

  % Add a text uicontrol to label the slider.
  txt = uicontrol('Style','text',...
     'Position',[400 45 120 20],...
     'String','Vertical Exaggeration');

  % Make figure visble after adding all components
  f.Visible = 'on';
  % This code uses dot notation to set properties.
  % Dot notation runs in R2014b and later.
  % For R2014a and earlier: set(f,'Visible','on');

  function setmap(source,event)
     val = source.Value;
     maps = source.String;
     % For R2014a and earlier:
```

```
        % val = get(source,'Value');
        % maps = get(source,'String');

        newmap = maps{val};
        colormap(newmap);
    end

    function surfzlim(source,event)
        val = 51 - source.Value;
        % For R2014a and earlier:
        % val = 51 - get(source,'Value');

        zlim(ax,[-val val]);
    end
end
```

The resulting UI displays a plot. Users can adjust the color map, change the vertical scaling, or clear the axes.